Intel® Itanium® 2 Processor

Specification Update

January 2004

Notice: The Intel® Itanium® 2 processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this specification update.

Document Number: 251141-020

Contents

Revision History 5

Preface 6

Summary Table of Changes..7

Identification Information 18

Errata (Processor and PAL) 21

Errata (IA-32 Execution Layer)..46

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Changes 50

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Clarifications 51

Itanium® 2 Processor (up to 3 MB L3 Cache) Documentation Changes 52

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Changes 53

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Clarifications 54

Itanium® 2 Processor (up to 6 MB L3 Cache) Documentation Changes 55

IA-32 Execution Layer Specification Clarifications 56

Revision History

Date	Version	Description
January 2004	020	Added errata 83-87; added Itanium 2 Processor (up to 3 MB L3 cache) PAL version 7.71 and Itanium 2 Processor (up to 6 MB L3 cache) PAL version 5.61; updated workaround for erratum 61. Updated problem and implication for IA-32 execution layer erratum 1; added IA-32 execution layer errata 2-16; added IA-32 Execution Layer Specification Clarifications 1-11.
December 2003	019	Added errata 80-82.
November 2003	018	Added errata 75-79.
October 2003	017	Added errata 71-74.
September 2003	016	Added errata 68-70; added Low Voltage Intel® Itanium® 2 Processor with 1.0 GHz with 1.5 MB L3 Cache and Intel® Itanium® 2 Processor with 1.40 GHz with 1.5 MB L3 Cache to Table 1-1; added S-Spec numbers SL76K and SL754; added *DP Optimized Intel® Itanium® 2 Processor Datasheet* to the list of Affected/Related Documents.
August 2003	015	Added errata 65-67; updated the *Intel® Itanium® Architecture Software Developer's Manual Specification Update* document number in the list of Affected/Related Documents.
July 2003	014	Added errata 61-62.
June 2003	013	Added Intel® Itanium® 2 processor with 6 MB L3 cache information; added new errata summary tables and Table 1-1; removed Specification Clarification 1; removed Documentation Changes 1-2; added errata 59, 63-64.
June 2003	012	Updated Implication for erratum 60.
June 2003	011	Added erratum 60; removed erratum 59.
May 2003	010	Added errata 55-59.
March 2003	009	Added errata 53-54; added PAL version 7.40.
February 2003	008	Updated workaround for erratum 48; added erratum 52; added PAL version 7.37.
January 2003	007	Added errata 49-51; added Documentation Change 2.
December 2002	006	Added errata 47-48.
November 2002	005	Added errata 43-46; added PAL version 7.36.
October 2002	004	Added errata 38-42.
September 2002	003	Added errata 30-37; added PAL version 7.31; added Documentation Change 1; added Specification Clarification 1.
August 2002	002	Added errata 20-29.
July 2002	001	Initial release of this document.

Preface

This document is an update to the specifications contained in the Affected/Related Documents table below. This document is a compilation of device and documentation errata, specification clarifications, and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

This document may also contain information that was not previously published.

Affected/Related Documents

Title	Document #
Intel® Itanium® 2 Processor Datasheet	250945
DP Optimized Intel® Itanium® 2 Processor Datasheet	253795
Intel® Itanium® 2 Processor Hardware Developer's Manual	251109
Intel® Itanium® Architecture Software Developer's Manual, Volume 1: Application Architecture	245317
Intel® Itanium® Architecture Software Developer's Manual, Volume 2: System Architecture	245318
Intel® Itanium® Architecture Software Developer's Manual, Volume 3: Instruction Set Reference	245319
Intel® Itanium® Architecture Software Developer's Manual Specification Update	248699
Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization	251110
Itanium® Processor Family System Abstraction Layer Specification	245359

Nomenclature

S-Spec Number is used to identify products. Products are differentiated by their unique characteristics, e.g. core speed, L3 cache size, package types, etc. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. These may cause the Itanium® 2 processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

Specification Changes are modifications to the current published specifications. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Note: Errata remain in the specification update throughout the product's lifecycle, or until a particular stepping is no longer commercially available. Under these circumstances, errata removed from the specification update are archived and available upon request. Specification changes, specification clarifications, and documentation changes are removed from the specification update when the appropriate changes are made to the appropriate product specification or user documentation (datasheets, manuals, etc.).

Summary Table of Changes

The following table indicates the errata, specification changes, specification clarifications, or documentation changes which apply to the Itanium 2 processors. Intel may fix some of the errata in a future stepping of the component or in a future release of the Processor Abstraction Layer (PAL), and account for the other outstanding issues through documentation or specification changes as noted. This table uses the notations indicated below.

Codes Used in Summary Table

Stepping/Version

X:	Errata exists in the indicated stepping, PAL version, or software extension. Documentation Change, Specification Change or Clarification that applies to this stepping.
(No mark) or (Blank box):	This erratum is fixed in listed stepping or specification change does not apply to listed stepping or PAL version.

Page

(Page):	Page location of item in this document.

Status

Doc:	Document change or update will be implemented.
Plan Fix:	This erratum may be fixed in a future stepping of the component, or in a future release of PAL.
Fixed:	This erratum has been previously fixed.
No Fix:	There are no plans to fix this erratum.

Row

|	Change bar to left of table row indicates this erratum is either new or modified from the previous version of this document.

Table 1-1. Definition Table

<table>
<tr><th>Processor</th><th>Abbreviation</th></tr>
<tr><td>Intel® Itanium® 2 Processor 900 MHz with 1.5 MB L3 Cache</td><td rowspan="2">Itanium 2 Processor (up to 3 MB L3 cache)</td></tr>
<tr><td>Intel® Itanium® 2 Processor 1.0 GHz with 3 MB L3 Cache</td></tr>
<tr><td>Low Voltage Intel® Itanium® 2 Processor 1.0 GHz with 1.5 MB L3 Cache</td><td rowspan="5">Itanium 2 Processor (up to 6 MB L3 cache)</td></tr>
<tr><td>Intel® Itanium® 2 Processor 1.40 GHz with 1.5 MB L3 Cache</td></tr>
<tr><td>Intel® Itanium® 2 Processor 1.30 GHz with 3 MB L3 Cache</td></tr>
<tr><td>Intel® Itanium® 2 Processor 1.40 GHz with 4 MB L3 Cache</td></tr>
<tr><td>Intel® Itanium® 2 Processor 1.50 GHz with 6 MB L3 Cache</td></tr>
</table>

Itanium® 2 Processor (up to 3 MB L3 Cache) Errata (Sheet 1 of 3)

No.	Processor Stepping	PAL Version							Pg.	Status	ERRATA
	B3	7.13	7.31	7.36	7.37	7.40	7.59	7.71			
1	X								21	No Fix	IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED does not count predicated off instructions
2	X								21	No Fix	Performance Monitor Interrupt raised when freeze bit is written to Performance Monitoring Counter register
3	X								21	No Fix	Priority agent requests with unit mask of I/O not counted
4	X								21	No Fix	Incorrect fault reporting on move to/from the RNAT or BSPSTORE application registers
5	X								22	No Fix	Power good deassertion affects boundary scan testing
6	X								22	No Fix	IA-32: CPUID instruction returns incorrect L3 cache size
7	X								22	No Fix	Performance Monitoring Event counters may be incorrect when using Instruction Address Range checking in fine mode
8	X								23	No Fix	Possible deadlock condition after ptc.g is issued on two-way system
9	X								23	No Fix	EPC, mov ar.pfs and br.ret instructions may combine to yield incorrect privilege level
10	X								24	No Fix	Removal of WAW hazard may lead to undefined result
11	X								24	No Fix	Unexpected data debug, data access or dirty bit fault taken after rfi instruction
12	X								25	No Fix	Incorrect privilege level may be granted if a failed speculation check precedes a privilege level change
13	X								25	No Fix	Floating-point instructions take a floating-point trap before Unimplemented Instruction Address trap
14		X							25	Fixed	PAL_MC_ERROR_INFO does not return an address for certain double bit ECC memory errors
15		X							25	Fixed	PAL_CACHE_READ and PAL_CACHE_WRITE return incorrect status for L1I cache access
16		X							26	Fixed	Unpredictable behavior if the system is awakened from low power mode by an MCA
17		X							26	Fixed	The system may lose an interrupt when SAL_CHECK reads the IVR
18		X							26	Fixed	A bus MCA nested within a recoverable or firmware-corrected bus MCA may not be handled correctly
19		X							26	Fixed	PAL reset sequence performed after a recovery check may result in incorrect system behavior
20		X							27	Fixed	PAL_HALT_LIGHT_SPECIAL provides PAL_HALT functionality
21		X							27	Fixed	PAL_TEST_PROC may access memory with the UC attribute
22	X								27	No Fix	L2 single bit data error promoted to MCA continues to flag a CMCI
23		X							27	Fixed	PAL_TEST_PROC requires specific tests be performed for correct operation
24		X							27	Fixed	PAL_TEST_INFO may return incorrect data for invalid test parameters
25		X							28	Fixed	PAL_CACHE_INIT may not function properly if levels of the cache hierarchy are specified

Itanium® 2 Processor (up to 3 MB L3 Cache) Errata (Sheet 2 of 3)

No.	Processor Stepping	PAL Version							Pg.	Status	ERRATA
	B3	7.13	7.31	7.36	7.37	7.40	7.59	7.71			
26		X							28	Fixed	PAL_SET_TIMEOUT may have an unexpected result when time-out = 0
27		X							28	Fixed	Concurrent MCAs that signal a BERR may not set PSP.bc correctly
28		X							28	Fixed	PAL_PLATFORM_ADDR may return an error if bit 63 is set
29		X							28	Fixed	PAL_TEST_PROC may overwrite predicate registers
30		X							28	Fixed	Recovery check fails if PAL_B is not found
31		X							29	Fixed	PAL procedure calls may have unexpected results if an incorrect PAL_B version is used
32		X							29	Fixed	Late self-test may have unexpected results during concurrent processor tests
33		X							29	Fixed	PAL_TEST_PROC may cause unexpected system behavior
34		X							29	Fixed	PAL halt procedures may overwrite predicate registers
35	X								30	No Fix	Two resets may be necessary to leave TAP test mode
36	X								30	No Fix	IA-32 instruction pointers may be overwritten under certain boundary conditions
37		X							30	Fixed	Initialization and ETM recovery may overwrite branch register
38			X						30	Fixed	PAL procedures may not save predicate register 3
39		X	X						30	Fixed	PAL_CACHE_INFO procedure may return undefined value
40			X						31	Fixed	PAL_HALT_LIGHT procedure may generate a spurious Performance Monitor Interrupt
41		X	X						31	Fixed	Unexpected system behavior after PAL_CACHE_FLUSH is executed
42		X	X						31	Fixed	PAL_TEST_PROC may not properly report self-test status
43	X								31	No Fix	PSR.ri may not reflect the correct slot upon entrance to the unimplemented address fault handler
44	X								32	No Fix	WC and WB memory attribute aliasing combine with FC and may cause processor live-lock
45	X								32	No Fix	Improper use of memory attribute aliasing may lead to out of order instruction execution
47	X								33	No Fix	Executing an rfi instruction that is located at the end of implemented physical memory can result in an unexpected unimplemented address fault
48	X								33	Fixed	IA-32: xchg instruction requires release semantics
49		X	X	X					33	Fixed	PAL MCA handler may not correctly set PSP.co bit
50		X	X	X					33	Fixed	PAL_MC_ERROR_INFO may return incorrect PSP information
51	X								34	No Fix	FPSWA trap may be missed
52	X								35	Fixed	WC evictions and semaphore operations combine to establish a potential live-lock condition
53	X								35	Fixed	The IA-32 cmpxchg8b instruction may not correctly set ZF flag
54		X	X	X	X	X	X	X	35	No Fix	PAL_TEST_PROC status return value
55	X								36	No Fix	Fault condition may generate incorrect address when using short format VHPT

Itanium® 2 Processor (up to 3 MB L3 Cache) Errata (Sheet 3 of 3)

No.	Processor Stepping	PAL Version							Pg.	Status	ERRATA
	B3	7.13	7.31	7.36	7.37	7.40	7.59	7.71			
57		X	X	X	X	X			36	Fixed	Cache snoops disabled on BINIT#
58	X								37	No Fix	RFI to UIA using single step mode may enter ss trap
60	X								37	No Fix	Specific instruction combination may disrupt subsequent operation
61	X								38	No Fix	IFS register may be invalidated during MCA or INIT
62							X		38	Fixed	Unimplemented memory access may occur while handling an INIT or MCA event
66		X	X	X	X	X	X	X	39	No Fix	PSP.cr is always set to zero (0) at PALE_INIT hand off to SALE_ENTRY
68		X	X	X	X	X	X		40	Fixed	Performance Monitoring Event counters may be incorrect after leaving a low-power state
69	X								40	No Fix	Instruction Breakpoint Register update may generate a false instruction debug fault
70	X								40	No Fix	Application fault may be missed on a br.ia instruction
71	X								41	No Fix	Machine check may not bring the system out of a low-power state
72		X	X	X	X	X	X		41	Fixed	Machine check event received during PAL execution may have unexpected results
73		X	X	X	X	X	X		41	Fixed	Rendezvous may result in spin loop due to incorrect rendezvous address passed to SAL
74		X	X	X	X	X	X		41	Fixed	Possible degradation in system performance when calling PAL_CACHE_FLUSH with int = 1 for certain cache memory types
75	X								42	No Fix	Memory read current transaction may fail to observe a st, ld.bias or lfetch.excl
76	X								42	No Fix	BINIT taken on 2x ECC and hard-fail errors with BINIT event signaling disabled
77	X								43	No Fix	Recoverable L3 cache tag ECC error may raise overflow error when CMCI are promoted to MCA
78	X								43	No Fix	L2 cache line with poison data results in unexpected fatal MCA
79	X								43	No Fix	XPN time-out with BINIT response disabled may cause system hang
80	X								43	No Fix	BINIT may be taken after a UC single byte access to ignored/reserved area of the Processor Interrupt Block
81	X								44	No Fix	Recoverable CMCI may combine with an L3 MCA error to cause fatal overflow error
82		X	X	X	X	X	X		44	Fixed	BERR may be indicated when the PAL MCA routine invalidates L2 cache lines
83		X	X	X	X	X	X	X	44	Plan Fix	Pending RSE interrupt during the PAL PMI handler may result in a system hang
84		X	X	X	X	X	X	X	44	Plan Fix	An INIT signaled during a PAL PMI flow may result in a system hang
85		X	X	X	X	X	X	X	44	Plan Fix	PMI serviced during the execution of PAL_MC_ERROR_INFO procedure may result in unpredictable processor behavior
86		X	X	X	X	X	X	X	45	No Fix	Data-poisoning bits not included in PAL_MC_ERROR_INFO cache_check and bus_check structures
87		X	X	X	X	X	X	X	45	Plan Fix	PAL_PREFETCH_VISIBILITY call not implemented

Itanium® 2 Processor (up to 6 MB L3 Cache) Errata (Sheet 1 of 2)

No.	Processor Stepping	PAL Version						Pg.	Status	ERRATA
	B1	5.37	5.61							
1	X							21	No Fix	IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED does not count predicated off instructions
2	X							21	No Fix	Performance Monitor Interrupt raised when freeze bit is written to Performance Monitoring Counter register
6	X							22	No Fix	IA-32: CPUID instruction returns incorrect L3 cache size
7	X							22	No Fix	Performance Monitoring Event counters may be incorrect when using Instruction Address Range checking in fine mode
8	X							23	No Fix	Possible deadlock condition after ptc.g is issued on two-way system
13	X							25	No Fix	Floating-point instructions take a floating-point trap before Unimplemented Instruction Address trap
22	X							27	No Fix	L2 single bit data error promoted to MCA continues to flag a CMCI
43	X							31	No Fix	PSR.ri may not reflect the correct slot upon entrance to the unimplemented address fault handler
45	X							32	No Fix	Improper use of memory attribute aliasing may lead to out of order instruction execution
47	X							33	No Fix	Executing an rfi instruction that is located at the end of implemented physical memory can result in an unexpected unimplemented address fault
54		X	X					35	No Fix	PAL_TEST_PROC status return value
55	X							36	No Fix	Fault condition may generate incorrect address when using short format VHPT
58	X							37	No Fix	RFI to UIA using single step mode may enter ss trap
59	X							37	No Fix	On Die Termination value does not meet specification
61	X							38	No Fix	IFS register may be invalidated during MCA or INIT
62		X						38	Fixed	Unimplemented memory access may occur while handling an INIT or MCA event
63	X							39	No Fix	JTAG Sample/Preload or EXTEST instruction usage
64	X							39	No Fix	CPU_CYCLES count includes data from halt states
65	X							39	No Fix	System bus signals can be driven while RESET# is asserted
66		X	X					39	No Fix	PSP.cr is always set to zero (0) at PALE_INIT hand off to SALE_ENTRY
67	X							40	No Fix	Incorrect Thermal Calibration Offset Byte value in the PIROM
69	X							40	No Fix	Instruction Breakpoint Register update may generate a false instruction debug fault
70	X							40	No Fix	Application fault may be missed on a br.ia instruction
71	X							41	No Fix	Machine check may not bring the system out of a low-power state
72		X						41	Fixed	Machine check event received during PAL execution may have unexpected results
73		X						41	Fixed	Rendezvous may result in spin loop due to incorrect rendezvous address passed to SAL
74		X						41	Fixed	Possible degradation in system performance when calling PAL_CACHE_FLUSH with int = 1 for certain cache memory types
75	X							42	No Fix	Memory read current transaction may fail to observe a st, ld.bias or lfetch.excl
76	X							42	No Fix	BINIT taken on 2x ECC and hard-fail errors with BINIT event signaling disabled

Itanium® 2 Processor (up to 6 MB L3 Cache) Errata (Sheet 2 of 2)

No.	Processor Stepping	PAL Version						Pg.	Status	ERRATA
	B1	5.37	5.61							
77	X							43	No Fix	Recoverable L3 cache tag ECC error may raise overflow error when CMCI are promoted to MCA
78	X							43	No Fix	L2 cache line with poison data results in unexpected fatal MCA
79	X							43	No Fix	XPN time-out with BINIT response disabled may cause system hang
80	X							43	No Fix	BINIT may be taken after a UC single byte access to ignored/reserved area of the Processor Interrupt Block
81	X							44	No Fix	Recoverable CMCI may combine with an L3 MCA error to cause fatal overflow error
82		X						44	Fixed	BERR may be indicated when the PAL MCA routine invalidates L2 cache lines
83		X	X					44	Plan Fix	Pending RSE interrupt during the PAL PMI handler may result in a system hang
84		X	X					44	Plan Fix	An INIT signaled during a PAL PMI flow may result in a system hang
85		X	X					44	Plan Fix	PMI serviced during the execution of PAL_MC_ERROR_INFO procedure may result in unpredictable processor behavior
86		X	X					45	No Fix	Data-poisoning bits not included in PAL_MC_ERROR_INFO cache_check and bus_check structures
87		X	X					45	Plan Fix	PAL_PREFETCH_VISIBILITY call not implemented

FPSWA Errata

No.	FPSWA Version						Pg.	Status	ERRATA
	1.09	1.12	1.18						
46	X						32	Fixed	FPSWA may not set the Denormal status flag correctly
56		X					36	Fixed	FPSWA version 1.12 may overwrite register fr12

IA-32 Execution Layer Errata

No.	IA-32 EL Version						Pg.	Status	ERRATA
	4.3.4277.4018								
1	X						46	No Fix	Ordering of loads and stores
2	X						46	No Fix	Segmentation not supported
3	X						46	No Fix	16-bit application mode not supported
4	X						46	No Fix	IA-32 floating-point state
5	X						47	No Fix	Floating-point C1 condition code flag support
6	X						47	No Fix	IA-32 floating-point pseudo-denormal, pseudo-NaN, and pseudo-infinity support
7	X						47	No Fix	Behavior of quiet and signaling NaNs
8	X						47	No Fix	IA-32 floating-point exceptions
9	X						47	No Fix	Partial support for EFLAGS
10	X						48	No Fix	EFLAGS and floating-point exception flag behavior
11	X						48	No Fix	RSM and IRET instructions raise incorrect faults
12	X						48	No Fix	Cross-modifying code
13	X						48	No Fix	Atomicity of lock-prefixed instructions making unaligned memory references
14	X						49	No Fix	Atomicity of lock-prefixed instructions making uncacheable memory references
15	X						49	No Fix	Noninterruptability of 32-bit unaligned and 16-byte stores
16	X						49	Fixed	IA-32 execution layer install and uninstall failures

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Changes

No.	Processor Stepping	PAL Version						Pg.	Status	SPECIFICATION CHANGES
	B3	7.13	7.31	7.36	7.37	7.40	7.59			
										None for this revision of the Specification Update

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Clarifications

No.	Processor Stepping	PAL Version						Pg.	Status	SPECIFICATION CLARIFICATIONS
	B3	7.13	7.31	7.36	7.37	7.40	7.59			
										None for this revision of the Specification Update

Itanium® 2 Processor (up to 3 MB L3 Cache) Documentation Changes

No.	Processor Stepping	PAL Version						Pg.	Status	DOCUMENTATION CHANGES
	B3	7.13	7.31	7.36	7.37	7.40	7.59			
										None for this revision of the Specification Update

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Changes

No.	Processor Stepping	PAL Version						Pg.	Status	SPECIFICATION CHANGES
	B1	5.37								
										None for this revision of the Specification Update

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Clarifications

No.	Processor Stepping	PAL Version						Pg.	Status	SPECIFICATION CLARIFICATIONS
	B1	5.37								
										None for this revision of the Specification Update

Itanium® 2 Processor (up to 6 MB L3 Cache) Documentation Changes

No.	Processor Stepping	PAL Version						Pg.	Status	DOCUMENTATION CHANGES
	B1	5.37								
										None for this revision of the Specification Update

IA-32 Execution Layer Specification Clarifications

No.	IA-32 EL Version						Pg.	Status	SPECIFICATION CLARIFICATIONS
	4.3								
1	X						56	Doc	Aliasing of MMX registers to FP registers
2	X						56	Doc	Floating-point and SSE precision
3	X						56	Doc	CPUID values represent the IA-32 execution layer processor model
4	X						56	Doc	IA-32 execution layer resides in the application virtual address space
5	X						56	Doc	Signal delivery may be postponed during code translation or garbage collection
6	X						56	Doc	Aborting threads could cause other process threads to hang
7	X						56	Doc	Core dump files cannot be produced correctly when an IA-32 process is aborted
8	X						57	Doc	The I/O Privilege Level (IOPL) mechanism is not implemented
9	X						57	Doc	Software interrupts must be supported by the OS
10	X						57	Doc	Intersegment calls require OS mechanism
11	X						57	Doc	Thread creation may be reported incorrectly to the OS

Identification Information

Intel® Itanium® 2 Processor Package Marking

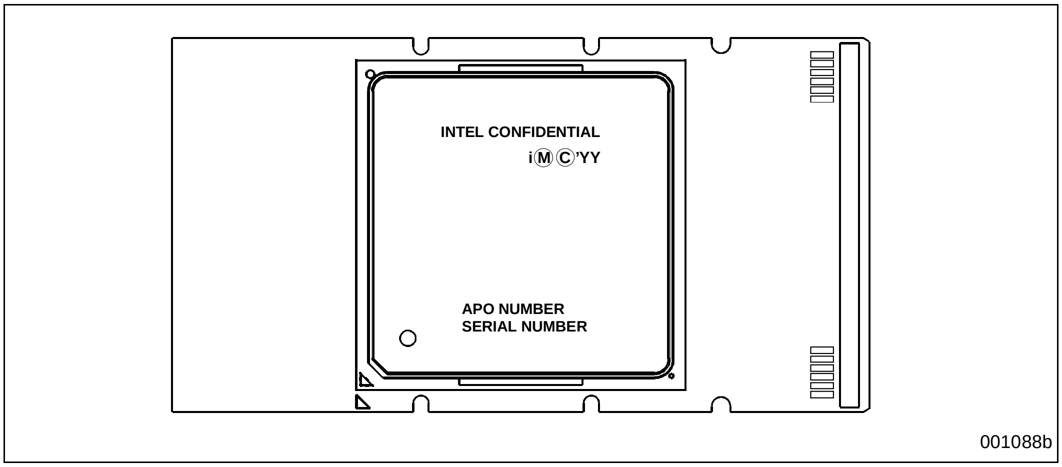
The following section details the processor top-side and bottom-side markings for the Itanium 2 processor and is provided as an identification aid. The processor top-side mark for the product is a laser marking on the Integrated Heat Spreader (IHS).

Processor Top-Side Marking

Figure 1-1 shows an example of the laser marking on the IHS. The processor top-side mark provides the following information:

- INTEL Brand/ INTEL Product
- Legal Mark
- Assembly Process Order (APO) Number
- Serial Number

Figure 1-1. Processor Top-Side Marking on IHS

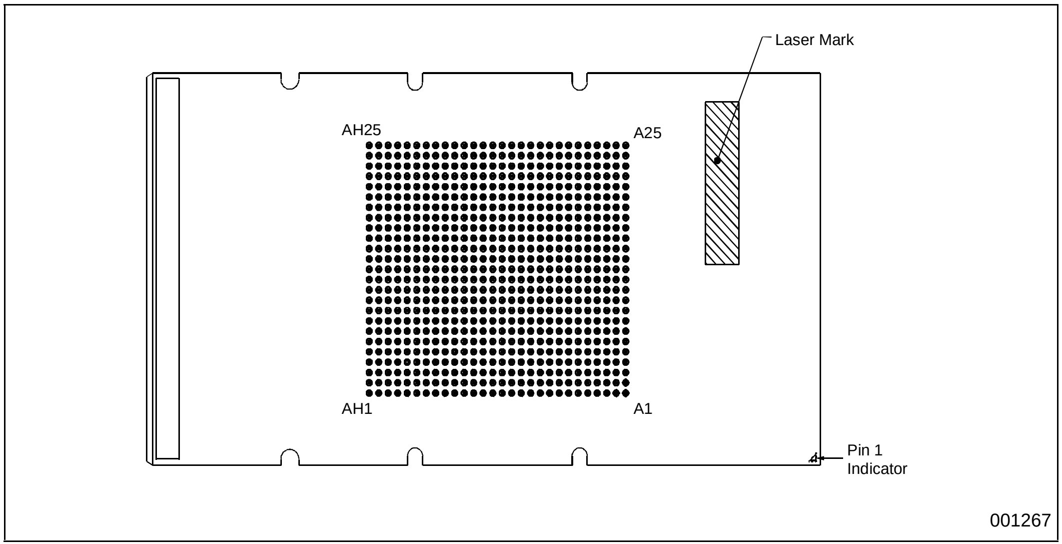


Bottom-Side Marking

The processor bottom-side mark for the product is a laser marking on the pin side of the interposer. Figure 1-2 shows the placement of the laser marking on the pin side of interposer. The processor bottom-side mark provides the following information:

- Product ID
- Finish Process Order (FPO) Number
- Serial Number
- S-Spec
- Country of Origin
- 2D Matrix Mark included on Itanium 2 processor (up to 6 MB L3 cache) only. Not included on Itanium 2 processor (up to 3 MB L3 cache).

Figure 1-2. Processor Bottom-Side Marking Placement on Interposer



Intel® Itanium® 2 Processor Identification and Package Information

S-Spec Number	Processor Stepping	CPUID[1]	Speed (MHz)	L3 Size (Mbytes)
SL67U	B3	001F000704h	1000/400	1.5
SL67V	B3	001F000704h	1000/400	3
SL67W	B3	001F000704h	900/400	1.5
SL6P5	B3	001F000704h	1000/400	1.5
SL6P7	B3	001F000704h	1000/400	3
SL6P6	B3	001F000704h	900/400	1.5
SL6XF	B1	001F010504h	1500/400	6
SL6XE	B1	001F010504h	1400/400	4
SL6XD	B1	001F010504h	1300/400	3
SL76K	B1	001F010504h	1400/400	1.5
SL754	B1	001F010504h	1000/400	1.5

1. The CPUID column in this table indicates the contents of bits 39:0 of CPUID Register 3. Bits 63:40 of this register are reserved. The Family ID for the Itanium® 2 processor is 0x1F.

Abbreviation	PAL Version[1]	Processor Stepping
Itanium® 2 Processor (up to 3 MB L3 cache)	7.13	B3
	7.31	B3
	7.36	B3
	7.37	B3
	7.40	B3
	7.59	B3
	7.71	B3
Itanium 2 Processor (up to 6 MB L3 cache)	5.37	B1
	5.61	B1

1. Please refer to the applicable PAL release notes for information regarding changes in each PAL release.

Errata (Processor and PAL)

1. IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED does not count predicated off instructions

Problem: The event monitor count for instructions retired (IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED) does not include the predicated off instructions.

Implication: The IA64_INST_RETIRED/IA64_TAGGED_INST_RETIRED performance monitoring events may report an incorrect count.

Workaround: Add the PREDICATE_SQUASHED_RETIRED event monitor count to the IA64_INST_RETIRED and/or the IA64_TAGGED_INST_RETIRED event monitor count to get the intended results.

Status: For the steppings effected, see the Summary Table of Changes.

2. Performance Monitor Interrupt raised when freeze bit is written to Performance Monitoring Counter register

Problem: The Performance Monitor Freeze (PMC[0].fr) bit within the Performance Monitoring Counter (PMC) register is used to stop performance event monitoring. This can be set by software or by an event counter overflow. Due to this erratum, the processor may raise a Performance Monitor Interrupt (PMI) when the freeze bit is set by software, even when the Performance Monitor Overflow Interrupt (PMC.oi) bit is not enabled and no overflow has occurred.

Implication: The processor may generate a PMI when it's not expected to do so.

Workaround: The interrupt service routine (ISR) needs to account for the spurious interrupt even if no performance monitor overflow is indicated.

Status: For the steppings effected, see the Summary Table of Changes.

3. Priority agent requests with unit mask of I/O not counted

Problem: The system bus allows for the BPRI# signal to be asserted one cycle before an ADS# is driven by the priority agent, provided no BREQ# pins are driven by the processor. Priority agent requests exhibiting this behavior are not counted by the system bus performance monitoring events when using a unit mask of 'I/O'.

Implication: The system bus performance monitoring events may report an incorrect count in this case.

Workaround: Measure the bus transactions for all bus masters (unit mask= 'ANY') and subtract from it the sum of the corresponding bus transactions on each local processor (unit mask= 'SELF').

Status: For the steppings affected, see the Summary Table of Changes.

4. Incorrect fault reporting on move to/from the RNAT or BSPSTORE application registers

Problem: Incorrect faulting behavior may be experienced under the following conditions:

1. A `mov.imm` (move immediate) to the ar.rsc register is executed in the same instruction bundle (or the next bundle with no intervening stop bits) as a mispredicted branch.
2. The mispredicted branch path includes another `mov.imm` to the same ar.rsc register, and is within two issue groups or less of the (mispredicted) branch instruction. This instruction is not executed. Also, the value moved to the rsc.mode field must be different than the value moved to rsc.mode in the `mov.imm` in step 1.

3. The correct branch path is then taken and includes a move to/from the ar.rnat or ar.bspstore registers, within the first bundle (or second bundle with no intervening stop bit) of the correct branch instruction.

Implication: When the above conditions line up (and there are no stalls or cache misses), the instruction in step 3 (move to/from ar.rnat or ar.bsp) uses the rse.mode value from the mov.imm in the mispredicted branch path instead of from instruction in step 1. As a result, there may be incorrect faulting behavior – an illegal opcode fault is missed (if rse.mode!= 0) or falsely indicated (if rse.mode = 0) and may result in inconsistent system behavior. This erratum has only been observed in a system validation environment.

Workaround: Use one of the following workarounds:

1. Use the register form of the move instruction or;
2. Ensure there is a stop bit between any `mov.imm` instruction to/from the ar.rsc registers and any subsequent branch instruction or;
3. Ensure that there is a stop bit between a “label” (branch target) and a subsequent move to/from ar.rnat/ar.bspstore.

Status: For the steppings affected, see the Summary Table of Changes.

5. Power good deassertion affects boundary scan testing

Problem: Deassertion of the PWRGOOD signal during boundary scan testing prevents the correct operation of the sampling functionality in the EXTEST and SAMPLE/PRELOAD JTAG commands.

Implication: As a result of this erratum the boundary scan chain function is disabled and will stop shifting data when the PWRGOOD signal is low.

Workaround: Keep the PWRGOOD signal asserted during boundary scan testing.

Status: For the steppings affected, see the Summary Table of Changes.

6. IA-32: CPUID instruction returns incorrect L3 cache size

Problem: The IA-32 `CPUID` instruction will always report the L3 cache size as 3 MB regardless of the actual size of the L3 cache.

Implication: IA-32 applications using the IA-32: `CPUID` instruction cannot rely on the cache size reported by this instruction. Native Itanium applications are not affected by this erratum and can access this information via the processor `CPUID` registers.

Workaround: Within the Linux* operating system (OS) environment, the ‘/proc/cpuinfo’ file contains this information. Within the Microsoft* OS environment this information is available through Windows API calls.

Status: For the steppings affected, see the Summary Table of Changes.

7. Performance Monitoring Event counters may be incorrect when using Instruction Address Range checking in fine mode

Problem: For performance monitoring events that use Instruction Address Range Matching set to ‘Fine Mode’ (PMC: 14, bit 13 = 1), the address matching capability will be inconsistent and may yield incorrect results.

Implication: Due to this erratum the results of an event counter while using ‘Fine Mode’ may not be correct.

Workaround: Use normal mode when using Instruction Address Range checking.

Status: For the steppings affected, see the Summary Table of Changes.

8. Possible deadlock condition after ptc.g is issued on two-way system

Problem: In a two processor system, a `ptc.g` instruction is issued on processor A. The execution of the `ptc.g` on processor A blocks the completion of a semaphore upon which processor B is waiting to become available. Concurrently processor B is issuing a long series of loads and stores with one or more instructions being retried or involves system memory access before being retired. Processor B's L2 cache entry queue, denoted as OzQ, is full and does not allow the `ptc.g` operation from processor A, entry into the L2 OzQ for completion. The `ptc.g` request will be presented again in three clock cycles. If processor B continues to execute a code sequence such that the L2 cache OzQ entries continue to be taken by other load/stores, then the `ptc.g` operation must continue to wait.

Implication: Due to this erratum, the system may deadlock while waiting for the `ptc.g` to be completed. Any break in, or completion of the code loop on processor B, including system interrupts, that allows the `ptc.g` operation to enter the L2 cache OzQ on processor B will be enough to release the deadlock condition. Additional processors will also change the time cycle necessary for this event to occur. This issue has only been observed during Random Instruction Testing in a system validation environment.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

9. EPC, mov ar.pfs and br.ret instructions may combine to yield incorrect privilege level

Problem: Due to certain internal timing and microarchitectural conditions, OS calls that return to user space from privilege code promote pages using a `br.ret` instruction, may not have the expected privilege level.

Using the following code sequence as an example:

```
<change of privilege level>   //epc on promote page; or br.ret
mov ar.pfs, [value];          //new pfs value has ppl < cpl
br.ret;;
```

In this case the `br.ret` is specified to not change the privilege level (`pl`) since the `br.ret` is asking to promote privilege to a numerically lower level. Current processor steppings may change current privilege level (`cpl`) to the `pl` at the beginning of the <change of privilege level>.

Implication: This erratum would result in having the `cpl` demoted and the user space application may not receive the correct privilege level. Privilege code promote page usage is limited and controlled by the OS. This issue has only been observed during random instruction testing in a system validation environment.

Workaround: Use one of the following workarounds:

1. Use an return from interrupt (`rfi`) instruction instead of `br.ret` to return from privilege code promote pages.
2. Insert a useless call-to-next bundle in all paths leading to a demoting `br.ret`.
3. PAL version 7.01 and above, have a workaround for this issue and it is enabled by default. The OS may implement one of the previous workarounds or a check mechanism, such that this PAL workaround can be disabled. Please review the PAL Release notes for details on the implementation of this workaround.

Status: For the steppings affected, see the Summary Table of Changes.

10. Removal of WAW hazard may lead to undefined result

Problem: Due to internal conditions an allowed WAW dependency may become a WAW hazard under the following circumstances:

- A move to the AR.PFS register is followed by a BR.CALL and both are executed in the same issue group, or
- A move to the AR.EC register is followed by a BR.RET and both are executed in the same issue group.

These combinations of instructions are legal WAW memory dependencies if one of the operations is predicated off. If preceding instructions (as indicated above) combine to change the predication on the BR.CALL or BR.RET from predicated true to predicated false, the processor may mistakenly decide the WAW hazard is still present and fail to recognize that the WAW has been removed which may result in an undefined value for ar.pfs or ar.ec.

The following code sequence demonstrates this issue:

```
p15 = 1;
;;
mov ar.pfs = R[x];
ld.c R[y] = [m]; //causes R[y] to be reloaded.
cmp.eq p15, p16 = R[y], R0;
(p15) br.call;
```

The RAW dependencies on `ld.c` to `cmp` and `cmp` to branch are legal. When the processor executes the issue group, the WAW hazard is present and the PFS results are undefined. If the `ld.c` misses the advanced load address table (ALAT), the `cmp` to branch will be re-executed, the new result of the `ld.c` causes the p15 value to change to false and thus eliminate the WAW. Then the processor may fail to recognize that the WAW has been removed.

Implication: An application may hang or signal an exception fault under these circumstances. The affected code sequence is not known by Intel to be generated in any current compiled code or exist in any current OS.

Workaround: Separate the predicate producing instruction from its consumer with a stop (as recommended in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 1: Application Architecture) or change the predication sequence to assure mutually exclusive predication of the instructions in the WAW dependency.

Status: For the steppings affected, see the Summary Table of Changes.

11. Unexpected data debug, data access or dirty bit fault taken after rfi instruction

Problem: A fault may be taken after a `rfi` instruction has been executed under the following circumstances. The IPSR.da or IPSR.dd bits are set to disable data debug/data access/dirty bit faults for the first Itanium processor system environment restore instruction. This is followed by an `rfi` instruction. The `rfi` instruction is followed by additional instructions that generate register stack engine (RSE) activity (`alloc, flushrs, br.ret`). The processor will see the RSE activity as valid Itanium system instructions and clear the ipsr.da/dd bits and this may result in an unexpected data debug, data access or dirty bit fault at the target of the `rfi`.

Implication: Due to this erratum an unexpected fault may be generated after an `rfi` instruction has been executed. This may slow the transition of the system into the Itanium system environment and log un-necessary errors.

Workaround: Separate the `rfi` from the RSE generating instruction by four issue groups of `nop` instructions.

Status: For the steppings affected, see the Summary Table of Changes.

12. Incorrect privilege level may be granted if a failed speculation check precedes a privilege level change

Problem: A failed speculation check instruction (`chk.s/chk.a/fchkf`) that is followed by a privilege change operation may result with the incorrect privilege level for instructions in the issue group of the privilege level change and beyond. The privilege changing instruction must occur within two clock cycles of the failed speculation check.

Implication: As a result of this erratum, the speculation check recovery code and subsequent instructions may have an incorrect privilege level.

Workaround: Do not use speculation near privilege changing instructions. The workaround for this erratum is to escalate failed speculation checks (speculation check re-steers) to the OS for recovery. This workaround is included in PAL version 7.01 and above.

Status: For the steppings affected, see the Summary Table of Changes.

13. Floating-point instructions take a floating-point trap before Unimplemented Instruction Address trap

Problem: A floating-point instruction that causes a floating-point trap and is the last instruction at the top of the physical address space should flag an Unimplemented Address trap before the floating-point trap.

Implication: The correct trap is flagged but only after the floating-point trap is taken first.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

14. PAL_MC_ERROR_INFO does not return an address for certain double bit ECC memory errors

Problem: PAL_MC_ERROR_INFO will report the address for the source of a double bit ECC memory error. However, under the conditions that the data with a 2x ECC error was prefetched to the L2 cache and later filled into the L1 cache, the source address will not be available.

Implication: PAL_MC_ERROR_INFO will not be able to report the address of a double bit ECC error in this case. Double bit errors that are consumed in this scenario will be not be recoverable.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

15. PAL_CACHE_READ and PAL_CACHE_WRITE return incorrect status for L1I cache access

Problem: The PAL_CACHE_READ and PAL_CACHE_WRITE procedures should return a status value of '–7' (which indicates this operation is not supported for this *cache_type* and *level*) when attempting to read or write to/from the L1I (instruction) and L1D (data) cache. When these procedures attempt to access the L1I cache an incorrect status value will be returned.

Implication: Due to this erratum, using these PAL procedures to access the L1I cache will result in the return of an incorrect status value, implying that the L1I cache is readable/writeable by these PAL procedure calls.

Workaround: Do not use these PAL procedures to access the L1D and L1I caches.

Status: For the steppings affected, see the Summary Table of Changes.

16. Unpredictable behavior if the system is awakened from low power mode by an MCA

Problem: If the system is in low power mode and an machine check abort (MCA), BERR# or BINIT# is signaled, the PALE_CHECK handler will be called to process the error condition. However, PALE_CHECK does not disable low power mode so that it can continue execution. As soon as PALE_CHECK attempts to drain the processor queues, the system may re-enter low power mode. This may cause incomplete handling of the error event and potentially, intermittent continuation of the same event during later signaled BINIT# events.

Implication: The processor can appear to be trapped in low power mode and/or system behavior may be unpredictable.

Workaround: Do not use low power mode or call the following PAL procedures: PAL_HALT, PAL_HALT_LIGHT or PAL_HALT_LIGHT_SPECIAL.

Status: For the steppings affected, see the Summary Table of Changes.

17. The system may lose an interrupt when SAL_CHECK reads the IVR

Problem: The PAL_REGISTER_INFO procedure returns an incorrect value to indicate that reading the Interrupt Vector Register (IVR), CR65 (Configuration Register 65) has no side effects. Based on this incorrect return value, when SAL_CHECK reads the IVR while saving system state data to NVRAM, a pending interrupt may be allowed to proceed before the current process has been completed.

Implication: The SAL_CHECK procedure relies on the return values of PAL_REGISTER_INFO to know which ARs and CRs are safe to read and save off. Due to this erratum, the SAL_CHECK reads the IVR, and consequently causes the corresponding bit in the IRR to be cleared and the ISR to change. The results of the interrupt routine currently being executed may be lost.

Workaround: After calling PAL_REGISTER_INFO with *info_request* = 3, System Abstraction Layer (SAL) can force the correct return value for CR65 by setting bit 1 of *reg_info_2* to a value of one.

Status: For the steppings affected, see the Summary Table of Changes.

18. A bus MCA nested within a recoverable or firmware-corrected bus MCA may not be handled correctly

Problem: During the processing of a non-fatal bus MCA, if a second bus MCA is received the second MCA may be missed.

Implication: A bus MCA received in this scenario may be missed and result in unpredictable system behavior. If the first MCA is fatal, system behavior remains correct.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

19. PAL reset sequence performed after a recovery check may result in incorrect system behavior

Problem: The PAL early self-test sequence performed after a recovery check may not properly serialize outstanding memory transactions.

Implication: As a result of this erratum, memory transactions that are outstanding at the point of transition from the recovery check handler to PAL may cause a deadlock condition and possibly hang the processor.

Workaround: SAL can call the PAL_MC_DRAIN procedure before returning to PAL from recovery check to ensure that outstanding transactions have completed.

Status: For the steppings affected, see the Summary Table of Changes.

20. PAL_HALT_LIGHT_SPECIAL provides PAL_HALT functionality

Problem: The PAL_HALT_LIGHT_SPECIAL procedure does not issue the stop grant acknowledge special bus cycle.

Implication: PAL_HALT_LIGHT_SPECIAL behavior will be the same as PAL_HALT.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

21. PAL_TEST_PROC may access memory with the UC attribute

Problem: The 'mem_attr' self-test in PAL_TEST_PROC may access memory with the UC attribute, even though the 'attributes' parameter does not allow UC access.

Implication: PAL_TEST_PROC may access uncacheable memory that may not be supported in some systems.

Workaround: Set bit 44 of the PAL_TEST_PROC procedure self-test control word (*st_control*) to '1' to skip the 'mem_attr' self-test.

Status: For the steppings affected, see the Summary Table of Changes.

22. L2 single bit data error promoted to MCA continues to flag a CMCI

Problem: With correctable machine check interrupt (CMCI) to MCA promotion enabled and an L2 single bit ECC data error occurs, an MCA is signaled but the CMCI continues to be raised. After the MCA is completed and the system calls the PAL_MC_RESUME procedure, a CMCI is raised if PSR.i = 1 (respond to external interrupts) and the CMCV.m = 0 (CMCI interrupts are pended).

Implication: A CMCI continues to be signaled on L2 1x ECC data errors, even if CMCI to MCA promotion is enabled.

Workaround: When enabling CMCI to MCA promotion, mask CMCIs by saving the state of CMCV.m then set CMCV.m = '1'. Restore the original state of CMCV.m when disabling promotion.

Status: For the steppings affected, see the Summary Table of Changes.

23. PAL_TEST_PROC requires specific tests be performed for correct operation

Problem: PAL_TEST_PROC self-test requires three specific tests be performed, otherwise the PAL procedure may report false failures or unexpected behavior.

Implication: The PAL_TEST_PROC procedure must perform the virtual hash page table (VHPT) test (bit 34), late floating-point test (bit 41) and RSE test (bit 45). Otherwise the system may have unexpected behavior or false test failures may be indicated.

Workaround: Bits 34, 41 and 45 in the PAL_TEST_PROC self-test control word (*st_control*) should be left at the default settings of '0' so these tests are performed.

Status: For the steppings affected, see the Summary Table of Changes.

24. PAL_TEST_INFO may return incorrect data for invalid test parameters

Problem: The PAL_TEST_INFO procedure may return incorrect data or status if the input arguments are not valid or are out of range for a given parameter.

Implication: Calling the PAL_TEST_PROC procedure with invalid inputs may result in incorrect data and/or status instead of indicating invalid arguments.

Workaround: Ensure that PAL_TEST_INFO input parameters are valid and within the argument's range.

Status: For the steppings affected, see the Summary Table of Changes.

25. PAL_CACHE_INIT may not function properly if levels of the cache hierarchy are specified

Problem: PAL_CACHE_INIT does not function properly when caches are selected individually.

Implication: A call to initialize the L1D cache may hang the processor and a call to initialize any other cache structure may fail and return an error.

Workaround: Call the PAL_CACHE_INIT procedure with level = –1 to initialize all caches.

Status: For the steppings affected, see the Summary Table of Changes.

26. PAL_SET_TIMEOUT may have an unexpected result when time-out = 0

Problem: Setting the input parameter time-out = 0 will disable the processor watchdog timer feature.

Implication: Calling PAL_SET_TIMEOUT with time-out = 0 disables the internal processor time-out function.

Workaround: Do not set the time-out parameter to ‘0’.

Status: For the steppings affected, see the Summary Table of Changes.

27. Concurrent MCAs that signal a BERR may not set PSP.bc correctly

Problem: In the case of concurrent MCAs that should result in BERR assertion, the PALE_CHECK handler may not set the PSP.bc (bus check error) bit before handing off to SAL.

Implication: As a result of this erratum, PAL_MC_ERROR_INFO will indicate that a bus error occurred, but the PSP at hand-off to SAL_CHECK will not.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

28. PAL_PLATFORM_ADDR may return an error if bit 63 is set

Problem: PAL_PLATFORM_ADDR should ignore bit 63 of the *address* argument. If this PAL procedure is called with bit 63 set to ‘1’ in the *address* argument, the procedure incorrectly returns status = –2 (invalid argument).

Implication: Due to this erratum, calling PAL_PLATFORM_ADDR with bit 63 of the address set to ‘1’ will return a status of ‘invalid argument’.

Workaround: Bit 63 should be set to ‘0’ when calling the PAL_PLATFORM_ADDR procedure to avoid this issue.

Status: For the steppings affected, see the Summary Table of Changes.

29. PAL_TEST_PROC may overwrite predicate registers

Problem: PAL_TEST_PROC may overwrite predicate registers pr4 and pr5, which should be preserved by the procedure.

Implication: PAL_TEST_PROC may modify pr4 or pr5, resulting in undefined behavior.

Workaround: Code calling this PAL procedure can save and restore these predicate registers around the PAL_TEST_PROC procedure.

Status: For the steppings affected, see the Summary Table of Changes.

30. Recovery check fails if PAL_B is not found

Problem: SAL may not be able to complete a recovery check when no PAL_B is present. The I/O port address, interrupt block and other features may not be available for SAL when recovery check is entered from PAL_A_SPEC.

Implication: Recovery check may fail if PAL_B is not available or is invalid.

Workaround: Ensure that the firmware interface table (FIT) entry for PAL_B points to a valid and correct version of PAL_B.

Status: For the steppings affected, see the Summary Table of Changes.

31. PAL procedure calls may have unexpected results if an incorrect PAL_B version is used

Problem: PAL procedures that call PAL_B may not provide the expected results if the first PAL_B entry in the FIT points to an incorrect version of PAL_B.

Implication: PAL procedures may fail if the PAL_B entry in the FIT is for an incorrect version.

Workaround: Ensure that the FIT entry for PAL_B points to the correct version.

Status: For the steppings affected, see the Summary Table of Changes.

32. Late self-test may have unexpected results during concurrent processor tests

Problem: While running PAL_TEST_PROC concurrently on more than one processor and the processors happen to access the same memory address space, a snoop may cause the ALAT test to fail.

Implication: If a processor self-test procedure is using the same memory space for concurrent processor testing, the ALAT test may fail and cause one processor to enter a spin loop.

Workaround: The ALAT test can be bypassed by setting bit 46 of the PAL_TEST_PROC self-test control word to ‘1’.

Status: For the steppings affected, see the Summary Table of Changes.

33. PAL_TEST_PROC may cause unexpected system behavior

Problem: The PAL_TEST_PROC ‘late floating-point load/store test’ may overwrite the fr2-fr5 and fr30-fr31 floating-point registers and the Bank 0 gr16–gr23 general registers may be overwritten by the ALAT, VHPT, translation lookaside buffer (TLB) and memory attributes tests.

Implication: PAL_TEST_PROC may corrupt the following registers: Bank 0 gr16–gr23 (general registers) and the fr2– fr5, fr30–fr31 (floating-point registers).

Workaround: Use different registers or save/restore the contents before/after running PAL_TEST_PROC.

Using the self-test control word of the PAL_TEST_PROC procedure, set the following bits to ‘1’: To avoid corrupting the Bank 0 general registers do not run the ALAT (bit 46), VHPT (bit 35), TLB (bit 33) and mem_attr (bit 44) tests. To avoid corrupting the floating-point registers do not run the late_fp_ld_st (bit 40) test.

Status: For the steppings affected, see the Summary Table of Changes.

34. PAL halt procedures may overwrite predicate registers

Problem: Predicate registers pr1, pr2 and pr3 may be overwritten by the PAL_HALT, PAL_HALT_LIGHT and PAL_HALT_LIGHT_SPECIAL procedures.

Implication: As a result of this erratum, pr1, pr2 and p3 may be overwritten.

Workaround: Save and restore the predicate registers, as needed when calling these PAL procedures.

Status: For the steppings affected, see the Summary Table of Changes.

35. Two resets may be necessary to leave TAP test mode

Problem: After accessing the test access port (TAP), issuing a RESET# may result in the processor entering an idle state instead of beginning normal operation. Signaling a second RESET# may be necessary to properly reinitialize the system under these conditions.

Implication: Due to this erratum, a second RESET# may be required to properly reinitialize the processor after the TAP port has been accessed. Normal system operation and boot process is not affected.

Workaround: Issue two resets to properly reinitialize the processor after accessing the TAP port.

Status: For the steppings affected, see the Summary Table of Changes.

36. IA-32 instruction pointers may be overwritten under certain boundary conditions

Problem: Under certain internal conditions involving branch prediction and multiple branch instructions, IA-32 instruction pointers may be overwritten and result in IA-32 instructions being executed out of order or incorrectly. An affected code sequence would have consecutive branch instructions that have started execution before being cancelled.

Implication: Due to this erratum, IA-32 instruction pointers may be overwritten resulting in incorrect IA-32 instruction execution.

Workaround: A workaround for this erratum is included in PAL version 7.31.

Status: For the steppings affected, see the Summary Table of Changes.

37. Initialization and ETM recovery may overwrite branch register

Problem: PAL INIT recovery code may overwrite br0, when it saves the system environment to the min-state save area. This erratum also affects the recovery path of an enhanced thermal management (ETM) alert that is generated while a system is in a low power mode.

Implication: INIT and ETM recovery code may overwrite br0, which prevents recovery with PAL_MC_RESUME and may result in unexpected system behavior.

Workaround: PAL version 7.31 fixes this issue.

Status: For the steppings affected, see the Summary Table of Changes.

38. PAL procedures may not save predicate register 3

Problem: The following PAL procedures may not properly save and restore predicate register pr3. The affected PAL procedures are:

PAL_CACHE_INIT, PAL_CACHE_LINE_INIT, PAL_CACHE_READ, PAL_CACHE_WRITE, PAL_CAR_INIT, PAL_COPY_INFO, PAL_COPY_PAL, PAL_PROC_SET_FEATURES, PAL_TEST_PROC

Implication: Predicate register 3 may be overwritten by the PAL procedures listed above.

Workaround: Save and restore pr3, as needed, when calling the aforementioned PAL procedures.

Status: For the steppings affected, see the Summary Table of Changes.

39. PAL_CACHE_INFO procedure may return undefined value

Problem: The PAL_CACHE_INFO procedure could return an invalid value in the config_info_1 '*at*' (cache memory attributes) field. When requesting information for the L2 and L3 cache, the '*at*' field may contain the value of 2, which is undefined.

Implication: The PAL_CACHE_INFO procedure, *cache memory attributes* field may return an undefined value.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

40. PAL_HALT_LIGHT procedure may generate a spurious Performance Monitor Interrupt

Problem: The PAL_HALT_LIGHT procedure may not properly set the value of the PMV.m bit on return from a low power state and as a result, a spurious PMI may be generated.

Implication: A spurious PMI may be indicated when using the PAL_HALT_LIGHT procedure.

Workaround: Set the PMV.m bit to ‘1’ (to mask PMIs) before calling PAL_HALT_LIGHT. Set the PMV.m bit to ‘0’ on return from the PAL_HALT_LIGHT procedure.

Status: For the steppings affected, see the Summary Table of Changes.

41. Unexpected system behavior after PAL_CACHE_FLUSH is executed

Problem: The PSR.ic bit is not restored after the PAL_CACHE_FLUSH procedure is executed with *cache_type* = 2. This may result in unexpected behavior when an interrupt is received after calling PAL_CACHE_FLUSH.

Implication: The system may not respond to interrupts as expected after PAL_CACHE_FLUSH is executed with *cache_type* = 2.

Workaround: Save and restore the PSR.ic bit as necessary, before and after calling the PAL_CACHE_FLUSH procedure.

Status: For the steppings affected, see the Summary Table of Changes.

42. PAL_TEST_PROC may not properly report self-test status

Problem: In the case that some PAL_TEST_PROC self-test functions fail, the *test_status* field may not indicate which self-test function has failed. Instead the failed test function may be raised as an initialization failure and the procedure will enter an infinite loop.

Implication: The PAL_TEST_PROC procedure may enter an infinite loop as a result of some failed self-tests, instead of operating in a functionally restricted manner.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

43. PSR.ri may not reflect the correct slot upon entrance to the unimplemented address fault handler

Problem: In the case of an `rfi` instruction that targets an instruction in slot 1 or 2 and the interrupt instruction pointer (IIP) points to an unimplemented physical address, the PSR.ri may point to slot 0 instead of slot 1 or 2 as expected. The required conditions to expose this erratum are: The processor is in physical address mode (PSR.it=0) and the IIP points to a physical memory address that is unimplemented.

Implication: When the processor attempts to execute on the indicated instruction bundle an unimplemented address fault will be taken and the restart instruction will indicate slot 0. Since no instruction in slot 0, 1, or 2 is executable under these conditions, there is no useful information lost when the unimplemented address fault is taken.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

44. WC and WB memory attribute aliasing combine with FC and may cause processor live-lock

Problem: Under certain conditions involving write coalescing (WC) stores and the execution of a flush cache (`fc`) instruction, the `fc` may not be able to proceed until the WC buffers have been emptied, resulting in a live-lock condition.

The live-lock is armed when one or more WC stores (st [A]) occur and allocate space in the processor's WC buffer. A store or load (st/ld [B]) with a writeback (WB) memory attribute is issued followed immediately by an `fc` (`fc[C]`) instruction. The `fc` is targeted to a virtual address with the same physical address as address [A], but with a WB memory attribute instead of WC. If address [B] shares the same physical address bits 14:7 with the flush cache target address [C], then the processor may live-lock.

Implication: This memory attribute aliasing (MAA) scenario is likely to occur for a short time in OS code page tear down or where a code page was previously accessed with the WC attribute, but is now implicitly considered to have WB attributes because memory translation has been disabled (PSR.dt=0).

Documented in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2, Section 4.4.11, as part of the process to properly transition to a new memory attribute, an `fc` instruction should be issued to flush the WC buffers. However, the text also states that a memory fence (`mf`) instruction should precede the `fc` instruction. Properly following this transition procedure will be sufficient to avoid the live-lock condition.

Workaround: Precede `fc` instructions with `mf` instructions where WC buffers may be non-empty.

Status: For the steppings affected, see the Summary Table of Changes.

45. Improper use of memory attribute aliasing may lead to out of order instruction execution

Problem: An `fc` instruction is issued to a virtual memory address that has been aliased as uncacheable (UC). This is immediately followed by a load/store to a WB memory address that points to same physical memory address that is targeted by the `fc`. Due to internal conditions, the load/store may be filled from the L2 cache rather then being filled from memory after the `fc` has been completed.

Implication: Using MAA in this manner requires the proper transitioning sequence as noted in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2, Section 4.4.11. Under these conditions, the order of operations observed directly on the system bus (by using a logic analyzer for example) may appear to be out of order, however there is no functional impact because the result of instruction execution will always be correct internally.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

46. FPSWA may not set the Denormal status flag correctly

Problem: In some cases when the Floating-Point Software Assistant (FPSWA) handles the following floating-point operation using the specified floating-point class/subclass types, the FPSWA may not return the correct Denormal/Unnormal (D) status flag setting in the Floating-Point Status Register (FPSR.sf0:8).

The affected operation is: Infinity * unnormalized number - Infinity = QNaN Indefinite.

Implication: As a result of this erratum, the FPSWA may indicate a Denormal/Unnormal exception fault where none has occurred.

Workaround: The FPSWA version 1.12 fixes this issue.

Status: For the steppings affected, see the Summary Table of Changes.

47. Executing an rfi instruction that is located at the end of implemented physical memory can result in an unexpected unimplemented address fault

Problem: Due to this erratum, when the processor is in physical mode and an `rfi` instruction at the end of physically implemented memory is executed, the processor will take an unimplemented address fault regardless of the real target of the `rfi` (IIP).

Implication: On a platform that supports the full 50 bits of physical address, under the above conditions an unexpected unimplemented address (UIA) fault could occur and the result depends upon the implementation of the UIA fault handler. This issue has only been observed in a pre-silicon simulation environment.

Workaround: Do not place an `rfi` instruction at the end of implemented physical memory.

Status: For the steppings affected, see the Summary Table of Changes.

48. IA-32: xchg instruction requires release semantics

Problem: The IA-32: `xchg` instruction can execute and write a value without it being explicitly ordered with respect to other IA-32 stores. The IA-32 memory model is strongly ordered and requires loads to have acquire (.acq) semantics and stores to have release (.rel) semantics to be executed in proper order. As a result of this requirement the `xchg` instruction requires the use of .acq and .rel semantics but only provides .acq semantics.

Implication: Due to this erratum, store operations may not be committed to memory in order with respect to IA-32 `xchg` operations.

Workaround: None at this time. PAL version 7.37 includes a fix for this issue.

Status: For the steppings affected, see the Summary Table of Changes.

49. PAL MCA handler may not correctly set PSP.co bit

Problem: The PAL MCA handler may not set the continuable bit (PSP.co) for potentially recoverable errors.

Implication: If the PSP.co bit is not set on recoverable errors, the OS and/or application may terminate when they could have potentially recovered from the error.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

50. PAL_MC_ERROR_INFO may return incorrect PSP information

Problem: When the PAL MCA handler has detected a fatal condition or has requested a SAL_MC_RENDEZ procedure call, the PSP returned from the PAL_MC_ERROR_INFO procedure may not contain all error information.

Implication: If SAL_CHECK is using the PSP returned from the PAL_MC_ERROR_INFO procedure call, some error information maybe missing which could result in application termination or a system hang.

Workaround: SAL_CHECK should use the PSP data at PALE_CHECK hand off rather than from PAL_MC_ERROR_INFO.

Status: For the steppings affected, see the Summary Table of Changes.

51. FPSWA trap may be missed

Problem: For Itanium 2 processor floating-point operations, when a *tiny*[1] result is computed (this usually corresponds to an underflow occuring), the processor should defer the computation to the FPSWA handler. In most cases, FPSWA will convert the result to a denormalized value that can be represented within the specified precision. However, for an extremely limited set of conditions, the processor fails to recognize this underflow and does not take the appropriate FPSWA trap.

Implication: Exposure to this issue occurs only under the following conditions:

1. Execution of one of the following instructions: `fma`, `fms`, `fnma`, `fpma`, `fpms`, `fpnma`.
2. The input operands for `fma`, `fms`, and `fnma` instructions (with or without.s or.d completers) must be capable of containing any combination of 64 bits in their significand, in register format. (If the significands of the operands are limited to less than 64 bits, the operation is not affected.)
3. The computed result is precisely $\pm$ 1.0 x $2^{(Emin-1)}$ [2]. This is a necessary (but not sufficient) condition as only an extremely small subset of the possible input operand combinations that generate a result of $\pm$ 1.0 x $2^{(Emin-1)}$ actually lead to a missed FPSWA trap. There must be a massive and specific cancellation generating the result prior to rounding to the destination precision.

For operations meeting these conditions, a small subset will not take the FPSWA trap. In these cases, the result ($\pm$ 1.0 x $2^{(Emin-1)}$) will not be representable within the floating-point format specified. For example, assuming single precision mode, the result would be $\pm$ 1.0 x 2^{-127}. Normally, the FPSWA handler converts this result to a denormalized value in the form of $\pm$ 0.1 x 2^{-126} to fit within the single precision exponent format. Without this conversion the following impacts may be observed:

- For `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=0 [3], the result in the register file is numerically correct and may be used for subsequent floating-point operations without issue. However, storing this value to memory (using stfs, stfd or stfe as appropriate) will result in a correctly signed zero instead of $\pm$ 0.1 x 2^{Emin}. This is equivalent to what occurs for the "Flush-To-Zero" (FTZ)[4] mode of operation.

  It is possible to preserve the correct numerical result (i.e. 1.0 x 2^{-127} for the single precision example above) by using the `stf.spill` instruction for stores and the `ldf.fill` instruction for any subsequent loads.
- For register precision `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=1, the result should be $\pm$ 1.0 x 2^{-65535}. However, the result in the register file will be $\pm$ 1.0 x 2^{-16382} in the form of a double-extended precision value.
- For parallel floating-point instructions (`fpma`, `fpms`, and `fpnma`), the result is stored in the register file as a correctly signed zero instead of $\pm$ 1.0 x $2^{(Emin-1)}$. Parallel floating-point instructions are not used in any known compiled code.

Workaround: For the vast majority of floating-point usage models, no workaround is recommended. The issue is limited to an extremely small subset of possible floating-point operations with a typical impact of replacing a tiny value ($\pm$ 1.0 x $2^{(Emin-1)}$) with a correctly signed zero. Any error due to this issue is typically less, in absolute value, than the majority of rounding errors that normally occur for floating-point operations. For applications requiring a workaround, the following actions are required:

---

1. A result is defined as tiny if it lies between -2^{Emin} and $+2^{Emin}$ after rounding to the destination precision with unbounded exponent range. Reference the *Intel® Itanium® Architecture Software Developer's Manual* or IEEE Standard 754-1985 for Binary Floating-Point Arithmetic for any additional clarifications.
2. For single precision, Emin = –126; for double precision, Emin = –1022; for double-extended precision, Emin = –16382; for register format, Emin = –65534.
3. Reference the *Intel® Itanium® Architecture Software Developer's Manual* for Floating-point Status Register (FPSR) bit definitions.
4. FTZ mode causes tiny results to be truncated to the correctly signed zero.

1. For `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=0, avoid input operands with 64-bit significands or use the `stf.spill` instruction for stores and the `ldf.fill` instruction for any subsequent loads.
2. Do not use register precision (FPSR.wre=1) for `fma`, `fms`, and `fnma` operations.
3. Do not use parallel floating-point operations (`fpma`, `fpms`, and `fpnma`).

Status: For the steppings affected, see the Summary Table of Changes.

52. WC evictions and semaphore operations combine to establish a potential live-lock condition

Problem: In the case that multiple processors are sharing memory space; when stores to WC memory are closely followed by semaphore operations to cacheable memory, the semaphore operations may block forward progress of the WC evictions. The semaphore will not be able to proceed until the WC stores are completed. As a result a live-lock condition is established between the WC evictions and the semaphore.

Implication: If the live-lock conditions are maintained, the system will eventually signal a BINIT. Other system activity or external interrupts may change availability of the system bus allowing the live-lock condition to be broken and the system will proceed as normal.

Workaround: None at this time. PAL version 7.37 includes a fix for this issue.

Status: For the steppings affected, see the Summary Table of Changes.

53. The IA-32 cmpxchg8b instruction may not correctly set ZF flag

Problem: The IA-32 `cmpxchg8b` instruction should set the Zero Flag (ZF) flag to 1 and update memory when the compare operation is successful. However, if due to memory contention, the upper four bytes (bits 63:32) of the targeted memory are changed during execution of the instruction and the lower four bytes remain unchanged, the ZF flag may be incorrectly set to 1, even though the upper four bytes of the compare are not equal.

Implication: If this erratum occurs, two processors in a multiprocessor environment can end up owning the same memory locations when there should be autonomous ownership.

The failing scenario can only occur in a multiprocessor system where there is heavy contention for the targeted memory location. It also requires that another processor manages to update only the upper four bytes of the targeted memory location during a very small timing window just prior to execution of the compare.

This erratum only affects the `cmpxchg8b` form of the IA-32 `cmpxchg` instruction and has only been observed in a synthetic test environment.

Workaround: PAL version 7.40 includes a fix for this erratum.

Status: For the steppings affected, see the Summary Table of Changes.

54. PAL_TEST_PROC *status* return value

Problem: The PAL_TEST_PROC procedure returns *status* = –3 when the call has completed successfully and some self-test errors have occurred. Normally –3 would indicate that the PAL procedure itself has failed.

Implication: SAL firmware that assumes self-test errors will be reported with *status* = 0 may not function correctly.

Workaround: When PAL_TEST_PROC returns *status* = –3, SAL should check the *self-test_state* to obtain more information about the self-test error and report the error.

Status: For the steppings affected, see the Summary Table of Changes.

55. Fault condition may generate incorrect address when using short format VHPT

Problem: A Debug Breakpoint or Protection Key fault may, under certain internal conditions, cause the physical address returned for a short format VHPT to not match the virtual address indicated by the VHPT entry.

The conditions under which this can occur are:

- The VHPT is enabled using the short format in a virtual addressing mode,
- Privilege level 0 access is available,
- Debug Breakpoint faulting is enabled (psr.db=1) and/or Protection Key Checking is enabled (psr.pk=1) and
- Certain cases of multiple TLB misses that result in multiple VHPT walks, where one of the VHPT walks is cancelled (because the faulting condition is removed) and then retried.

It is possible under these specific conditions that the short format data associated with the retried VHPT walk may be associated with another.

Implication: If this erratum were to occur, a Protection Key fault or an Instruction or Data Debug fault may cause a VHPT entry to be incorrect. This may result in an incorrect code sequence being executed and would leave the system in an indeterminate state.

With regard to Debug Breakpoint faulting, exposure is limited to development code environments only. In the case of Protection Key checking, there is no known exposure for all current operating systems as the conditions for this erratum are not met.

Workaround: This erratum affects only the short format VHPT, using the long format of the VHPT will avoid either of these faulting conditions. Additionally, in the case of Debug Breakpoint Faulting, prevent the DBR from ever matching any portion of the VHPT by checking the VHPT before allowing the DBR to be set.

Status: For the steppings affected, see the Summary Table of Changes.

56. FPSWA version 1.12 may overwrite register fr12

Problem: The FPSWA version 1.12 may overwrite register fr12 when handling FPSWA faults caused by the `fma`, `fms` and `fnma` instructions consuming denormalized or unnormalized values. FPSWA should only use registers fr6-fr11.

Implication: Operating systems are required to save and restore fr6-fr11 when handling FPSWA faults. Any operating system that also saves and restores additional registers including fr12 is not susceptible to this issue. Depending on how an application uses fr12 and how the operating system preserves it, this erratum could lead to a number of different failure scenarios including incorrect data. The only known current exposure is with the Linux* operating system. This erratum is limited to FPSWA version 1.12.

Workaround: Upgrade to FPSWA version 1.18 or later which corrects the issue.

Status: For the steppings affected, see the Summary Table of Changes.

57. Cache snoops disabled on BINIT#

Problem: After a BINIT# is signaled the processor will disable snoops to contain the propagation of any errors. The resulting MCA condition will cause the processor to enter the PAL MCA handler, which will invalidate the processor caches before the hand-off to SAL. The PAL MCA handler does not re-enable cache snoops before the hand-off to SAL.

Implication: This erratum only occurs after a BINIT event, thus any potential impact is limited to error handling after this fatal event. As a result of this issue cache coherency will not be maintained after a BINIT error. SAL code that runs uncacheable is unaffected. Cache coherency is restored after the processor is reset as part of the normal BINIT event handling.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

58. RFI to UIA using single step mode may enter ss trap

Problem: In single step mode, a single step trap may be incorrectly taken on an `rfi` instruction when the `rfi` attempts to address unimplemented memory.

Implication: The single step trap should not be taken on an `rfi` instruction. The result of this erratum would be an indication that the single step/`rfi` instruction was completed successfully before entering the unimplemented memory address (UIA) trap.

Workaround: Avoid taking an `rfi` to an UIA.

Status: For the steppings affected, see the Summary Table of Changes.

59. On Die Termination value does not meet specification

Problem: The value of the On Die Termination (ODT) does not meet the specified range of 45 Ohms ± 15% when measured at Vol. The actual value is 37 Ohms ± 5% when measured at Vol. At output voltages above 0.6V, the ODT values are within the correct range.

Implication: The stronger value of ODT could result in a higher output low voltage (Vol) and reduced noise margins. Measurements on an Intel platform have not shown any noticeable increase in Vol and noise margins are within specified ranges.

Workaround: This erratum does not affect any system using on-board termination. No workaround is recommended for platforms using ODT in a 3-load configuration. ODT termination is not recommended for 5-load bus configurations, those should use on-board termination.

Status: For the steppings affected, see the Summary Table of Changes.

60. Specific instruction combination may disrupt subsequent operation

Problem: A specific combination of memory and integer instructions may cause the result of a prior integer operation to be incorrect. The combination of instructions necessary for the failure is:

1. Four or more arithmetic and at least one additional operation executing concurrently, immediately followed by a subsequent integer operation that consumes data from the previous operation.
2. Particular data patterns are also required.
3. This erratum is more likely at higher temperatures and higher processor core speeds.

Implication: As a result of this erratum, an integer operation may consume incorrect data leading to unpredictable system behavior. In some instances, a fatal DTLB MCA or memory page fault may occur.

Workaround: Intel recommends implementing one of the following workarounds:

- Reduce the processor operating frequency to 800 MHz by adjusting the system bus ratio to 2:8. Consult the *Intel® Itanium® 2 Processor Hardware Developer's Manual* for complete information on setting the system bus ratio.
- Avoid use of the susceptible code sequence and/or add stops between affected instruction groups.

61. IFS register may be invalidated during MCA or INIT

Problem: If an interrupt service routine (ISR) is reading the interruption function state (IFS) control register when the processor detects an MCA or receives an INIT event, under certain internal timing conditions the destination register of the IFS read may indicate that the IFS is invalid.

To be exposed to this issue the processor must be in the proper context to read the IFS control register. This requires executing at privilege level 0, having interruption collection disabled (psr.ic=0), and the IFS register must be valid (ifs.v=1). Executing a `cover` instruction sets ifs.v=1. In addition MCAs and INITs must not be masked (psr.mc=0).

Implication: When the ISR issues a `rfi` instruction, the return value of current frame marker (CFM) may not be properly restored. The contents of the backing store application registers may not be correct in this situation. Indeterminate system operation can result if this erratum occurs.

Workaround: PAL version 7.59 for the Itanium 2 processor (up to 3 MB L3 cache) and PAL version 5.37 for the Itanium 2 processor (up to 6 MB L3 cache) contain a workaround that corrects the possible problem when reading the IFS control register. This workaround requires the OS to abide by some specific restrictions. All known current OS releases adhere to these restrictions. These restrictions are:

1. There are no branches within a small window of code after the IFS read. The length of this window is the shorter of either three bundles or two instruction groups.
2. A `cover` instruction must not be followed by a branch to a bundle within the window after the IFS read. The window is as defined in item #1.
3. All ISR code from the `cover` instruction to the earlier of either changing psr.ic to 1 or the `rfi` at the end of the ISR, must exist within the same contiguous region of physical memory.
4. A `bsw.1` instruction must not be used within the ISR after a `cover` instruction and prior to the IFS read. This applies only if the destination register of the `mov` from IFS is r29, r30, or r31. PAL version 7.71 for the Itanium 2 processor (up to 3 MB L3 cache) and PAL version 5.61 for the Itanium 2 processor (up to 6 MB L3 cache) removes the requirement for this restriction.
5. After an MCA or INIT event, if this workaround is unable to properly recover the IFS control register state, a fatal MCA will be signaled to prevent unpredictable machine behavior.

Status: For the steppings affected, see the Summary Table of Changes.

62. Unimplemented memory access may occur while handling an INIT or MCA event

Problem: This erratum involves possible incorrect behavior if an ISR or fault handler exists in physical memory near address zero. If such an ISR or fault handler is executing and a `cover` instruction has been executed (IPSR.ic=0, IFS.v=1) and then an INIT or MCA event occurs while the handler is within the address range of 0 to 0x20, the processor can incorrectly access unimplemented memory. This results in a second MCA generated by the incorrect PAL behavior and this MCA occurs while interruption collection (IPSR.ic=0) is disabled.

Implication: It is highly unusual that any part of an ISR or fault handler including the `cover` instruction would be located in the first few locations of physical memory. Current known OS releases are not affected. If this erratum were to occur, receiving nested MCAs is not a condition the OS expects to encounter. A system crash or fatal error event may occur.

Workaround: Do not locate ISR or fault handling code with a `cover` instruction within the physical address range of 0 to 0x20.

Status: For the steppings affected, see the Summary Table of Changes.

63. JTAG Sample/Preload or EXTEST instruction usage

Problem: When using the JTAG Sample/Preload or EXTEST boundary scan instruction, all internal signals in the BSDL file must have their safe values loaded into the boundary scan serial data register when the JTAG state machine enters the update DR state. Failure to do so will result in putting the component into a non-operational test mode.

Implication: Failure to load the data register with safe values for all internal signals contained in the BSDL file may result in putting the part into a non-operational test mode.

Workaround: When loading the JTAG data register during the Sample/Preload instruction, or EXTEST instruction, load safe values contained for all internal signals contained in the BSDL files.

Status: For the steppings affected, see the Summary Table of Changes.

64. CPU_CYCLES count includes data from halt states

Problem: The event monitor count for CPU_CYCLES accumulates the count of elapsed processor clock cycles even in a light halt state. The CPU_CYCLES counter is not expected to accumulate the count when the processor is in a light halt or powered down state.

Implication: The CPU_CYCLES performance monitoring event may report an incorrect count if the processor goes into a light halt state.

Workaround: PAL 5.37 contains a workaround for this erratum.

Status: For the steppings affected, see the Summary Table of Changes.

65. System bus signals can be driven while RESET# is asserted

Problem: Upon the first assertion of RESET# after PWRGOOD is asserted, the processor may drive some of the system bus signals. The processor should tristate all system bus signals within two bus clocks of the assertion of RESET#. Due to this erratum, the processor may not tristate all system bus signals within this two clock limit.

Implication: The system bus state during this initial time window with RESET# asserted cannot be determined. Since no processor execution takes place with RESET# asserted, this does not affect processor operation after the RESET# sequence has been completed.

Workaround: The state of the system bus signals during the initial RESET# sequence should be ignored.

Status: For the steppings effected, see the Summary Table of Changes.

66. PSP.cr is always set to zero (0) at PALE_INIT hand off to SALE_ENTRY

Problem: When PALE_INIT completes the PAL handling of an initialization (INIT) event, status information is indicated in the Processor State Parameter (PSP) register at the hand off to SALE_ENTRY. After any INIT event, the state of PSP.cr (bit 20) will incorrectly be set to zero (0) which indicates that the control registers are not valid. This erratum only pertains to the state of the PSP.cr bit, the actual contents of all control registers after the INIT is correct and the control register information recorded by PALE_INIT in the min-state save area is also correct.

Implication: Based on the incorrect state of the PSP.cr bit, the control register information recorded in the min-state save area could be assumed to be invalid. In fact, the information is an accurate recording of the control register states at the time of the INIT event. Furthermore, the control registers are valid at the PALE_INIT to SALE_ENTRY hand off.

Workaround: The value of PSP.cr can be assumed to be one (1) (valid) after any INIT event.

Status: For the steppings effected, see the Summary Table of Changes.

67. Incorrect Thermal Calibration Offset Byte value in the PIROM

Problem: The Thermal Calibration Offset Byte value in the PIROM was incorrectly programmed to eight (8). The correct value for the Thermal Calibration Offset Byte should be zero (0).

Implication: Systems using the Thermal Calibration Offset Byte value programmed in the PIROM may report inaccurate information for the following:

1. Temperature readings from the SMBus.
2. Upper and lower thresholds for THRMALERT#.

Workaround: Systems should use a value of 0 for the Thermal Calibration Offset Byte.

Status: For the steppings effected, see the Summary Table of Changes.

68. Performance Monitoring Event counters may be incorrect after leaving a low-power state

Problem: On entry into the PAL_HALT_LIGHT procedure the performance monitoring counters that are expected to continue monitoring events in a low-power state will be frozen until the processor returns to full power.

Implication: As a result of this erratum, the Performance Monitoring Event counters noted in Section 10.3.11 of the *Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization* may be incorrect after leaving a low-power state.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

69. Instruction Breakpoint Register update may generate a false instruction debug fault

Problem: An incorrect instruction debug fault may be indicated on a write to the enable and mask bits in the Instruction Breakpoint Registers (IBR).

Implication: Code execution may fault on the false instruction debug fault generated by either the write into the IBR or on other instructions depending upon how the debug mask bits have been set. The IBR is only accessible in privilege level 0. OS software debug tools may or may not use this debug breakpoint feature.

Workaround: Disable Debug Breakpoint Faulting (Psr.db=0) before writing the enable and mask bits in the IBR and then re-enable Debug Breakpoint Faulting.

Status: For the steppings effected, see the Summary Table of Changes.

70. Application fault may be missed on a br.ia instruction

Problem: An Illegal Operation Fault may not be indicated when executing the `br.ia` instruction and the BSPSTORE register is not equal to the BSP register.

Implication: An Illegal Operation Fault should be indicated if an unconditional branch (`br.ia`) into IA-32 application space is made without first issuing a Flush Register Stack (`flushrs`) instruction to ensure that BSP and BSPSTORE are equal and the register stack partitions are saved. As a result of this erratum it is possible that the IA-32 application code will begin execution before indicating a fault.

Workaround: Ensuring that a `flushrs` instruction is issued before executing the `br.ia` instruction, as required by the *Intel® Itanium® Architecture Software Developer's Manual*, will eliminate the exposure to this erratum.

Status: For the steppings effected, see the Summary Table of Changes.

71. Machine check may not bring the system out of a low-power state

Problem: In the case that the processor has entered a low-power state and MCA checking is masked (PSR.mc=1) a machine check event may not bring the processor out of the low-power state.

Implication: The *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2 (Document No. 245318) documents that the processor should return to the Normal state upon receipt of an unmasked external interrupt, machine check, Reset, PMI or INIT. As a result of this erratum a machine check event received in a low-power state while machine check aborts are being masked, will not be serviced until the system is returned to a normal operating state by any other wakeup event.

Workaround: Enable machine check abort checking (PSR.mc=0) before entering a low-power state.

Status: For the steppings effected, see the Summary Table of Changes.

72. Machine check event received during PAL execution may have unexpected results

Problem: Depending on internal conditions, a machine check event (MCA) received during the execution of certain PAL procedures may have unexpected results.

Implication: During the execution of the following PAL procedures; PAL_CACHE_FLUSH, PAL_CACHE_INIT, PAL_CACHE_LINE_INIT, PAL_CACHE_READ, PAL_CACHE_WRITE, PAL_CAR_INIT, PAL_TEST_PROC and PAL_VM_TR_READ, if an MCA event is received the PAL procedure may fail. Depending on when the MCA is received and the execution environment, the results may range from a PAL or system error to a processor hang. In most cases the procedure will execute correctly.

Workaround: Ensure that machine check abort checking is disabled (PSR.mc=1) before calling the PAL procedures noted above.

Status: For the steppings effected, see the Summary Table of Changes.

73. Rendezvous may result in spin loop due to incorrect rendezvous address passed to SAL

Problem: When the PAL determines that an error has occurred which could cause a multiprocessor system to lose error containment, it must rendezvous the other processors in the system before proceeding with further processing of the machine check. This is accomplished by branching to SAL with a non-zero return vector address. It is then the responsibility of the SAL to rendezvous the other processors and return to PAL through this return address. It is possible for PAL to pass an incorrect return address to SAL during the hand off for processor Rendezvous.

Implication: The normal mode of operation during a rendezvous event is a blue screen, while the processors enter a spin loop. As a result of this erratum, the hand off to SAL may be fatal.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

74. Possible degradation in system performance when calling PAL_CACHE_FLUSH with int = 1 for certain cache memory types

Problem: When the PAL_CACHE_FLUSH procedure is called with int = 1, external interrupts will be polled periodically while the specified cache type(s) are being flushed. If an external interrupt is seen, this procedure will return and allow the caller to service the interrupt before all cache lines in the specified cache type are flushed. The problem is that when PAL_CACHE_FLUSH is called again to resume the flush operation from where it was interrupted, PAL attempts to start the flush operation over again rather than continuing from the point of interruption. This erratum affects *cache_types* 1, 2, and 3 as described in the The *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2 (Document No. 245318).

Implication: If additional interrupts continue to occur before the completion of the PAL_CACHE_FLUSH, the procedure may never complete. This may result in degraded system performance due to one processor not being available or appearing to be stalled. This issue has only been observed in a validation test environment.

Workaround: Do not call the PAL_CACHE_FLUSH procedure with int = 1 and *cache_type* = 1, 2 or 3.

Status: For the steppings effected, see the Summary Table of Changes.

75. Memory read current transaction may fail to observe a st, ld.bias or lfetch.excl

Problem: A memory read current transaction allows a chipset to access a coherent copy of a cache line in a caching agent without affecting the cache line state in the caching agent. This transaction avoids later cache misses and subsequent transactions by the cache agent to again cache the line.

The erratum requires the following code sequence:

1. Given two addresses X and Y, which would map to two different L2 cache lines:
   a. A memory read current (same cache line as X) must occur coincident to the sequence: load(X)... load (same cache line as X)... store (same cache line as X);
   or
   b. A memory read current (same cache line as X) must occur coincident to the sequence: load(X)... semaphore (Y)... store (same cache line as X);
   or
   c. Either of the above where store(X) is replaced with an ld.bias(X) or an lfetch.excl(X).
2. First `load(X)` need not be cached but has to fill the L2 to an E-state.

If systems utilize the memory read current transaction and execute the above code sequence, and specific internal micro-architectural timings are met, the cache line may be updated to an incorrect state by the processor.

Implication: Usage models are not known to exist where the *st*, *`ld.bias`* or *`lfetch.excl`* to a cache line (X) at or near the time of a memory read current transaction targeting cache line (X). If the conditions as described are met, a future external access to the memory contained in cache line (X) will not receive the expected *hitm* snoop response from the processor. Internal accesses will miss and be issued to the system interface.

Workaround: Memory read current transactions should not be used in situations where the above conditions are met.

Status: For the steppings effected, see the Summary Table of Changes.

76. BINIT taken on 2x ECC and hard-fail errors with BINIT event signaling disabled

Problem: A Bus Initialization (BINIT) event may still be signaled after a multiple-bit ECC or hard-fail error, even if BINIT event signaling/checking is disabled.

Implication: Multiple bit ECC errors, PTC and IPI operations that experience transactions errors may normally signal a Machine check that result in a BINIT response. However, when the BINIT response is disabled a BINIT is not expected. As a result of this erratum a BINIT will still be signaled for these types of errors even with the BINIT response disabled.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

77. Recoverable L3 cache tag ECC error may raise overflow error when CMCI are promoted to MCA

Problem: In the case that CMCIs are promoted to MCA, certain internal conditions combine with an L3 cache tag ECC error to indicate an overflow error and signal a fatal MCA.

Implication: An L3 cache tag ECC error is normally a recoverable CMCI but when CMCIs are being promoted to MCA, the error is promoted as a fatal MCA event instead of being firmware corrected. The fatal MCA is indicated if the cache line tags are snooped after the ECC error is flagged but before the MCA is taken.

Workaround: A workaround is under investigation.

Status: For the steppings effected, see the Summary Table of Changes.

78. L2 cache line with poison data results in unexpected fatal MCA

Problem: An L2 cache line with latent 2x ECC or poisoned data that is snooped before being consumed may incorrectly signal a fatal MCA.

Implication: An L2 cache line with a 2x ECC or an error that results in a cache line being poisoned should indicate a CMCI unless the data is consumed by a processor. A subsequent snoop hit to the poisoned cache line may cause the errant line to be flagged as an error twice, which would result in a machine check overflow and a fatal MCA being taken rather than a CMCI. This erratum does not apply to consumed poisoned data

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

79. XPN time-out with BINIT response disabled may cause system hang

Problem: In the case where the BINIT response to a processor internal time-out response is disabled, a second XPN time-out error may result in a system hang.

Implication: If an XPN time-out occurs such that a BINIT should be taken but is not due to the fact that the BINIT on an internal time-out response has been suppressed. A second XPN time-out error may result in the system hanging because the time-out counter was not reset after the first internal time-out.

Workaround: Do not suppress the BINIT response to a processor internal time-out.

Status: For the steppings effected, see the Summary Table of Changes.

80. BINIT may be taken after a UC single byte access to ignored/reserved area of the Processor Interrupt Block

Problem: A Bus INITialization (BINIT#) event may be signaled after an uncacheable (UC) single byte access to any ignored/reserved area in the upper half of the Processor Interrupt Block.

Implication: Unsupported accesses result in undefined behavior of the processor, hence the BINIT# response is taken to re-establish a consistent execution environment. In other cases the unsupported access can be ignored. Single byte UC access to the ignored or reserved areas of the IPI block should be ignored but as a result of this erratum a BINIT# is signaled.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

81. Recoverable CMCI may combine with an L3 MCA error to cause fatal overflow error

Problem: In the case where a recoverable L3 cache or system bus error flags a Correctable Machine Check Interrupt (CMCI) and is followed by specific MCA events, the overflow bit may be set and result in a fatal error. The specific MCA events are a L3 cache, system hard-fail, local BINIT# or a non-coherent UC/WC memory access that receives a HITM response.

Implication: As a result of this erratum a CMCI or MCA event that is normally recoverable, if supported by the OS, may set the overflow bit and signal a global BINIT#.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

82. BERR may be indicated when the PAL MCA routine invalidates L2 cache lines

Problem: A Bus ERRor (BERR#) may be signaled when a read hit occurs to the same L2 cache line that a PAL MCA routine is in process of invalidating.

Implication: As a result of this erratum a BERR# may be signaled after a hard-fail error, if a read hits a cache line while the line is being invalidated via the MESI protocol tags but before the cache line ECC has been updated.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

83. Pending RSE interrupt during the PAL PMI handler may result in a system hang

Problem: A system hang may be the result of a pending RSE interruption during the execution of the PAL PMI handler.

Implication: Depending on the execution of the PAL PMI flow and a pending RSE interruption, the result may be unsuccessful handling of the PAL PMI handler which would lead to a system hang.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

84. An INIT signaled during a PAL PMI flow may result in a system hang

Problem: If an MCA/INIT is signaled during the execution of the PAL PMI handler where an `rfi` is in the instruction pipeline but not yet executed, the system may hang as the `rfi` is aborted before returning from the MCA/INIT procedure.

Implication: There is a small window of exposure where the `rfi` can be in the instruction pipeline and an MCA/INIT is taken, where it aborts the `rfi` before the `rfi` has been executed. If these conditions are met the result may be a system hang.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

85. PMI serviced during the execution of PAL_MC_ERROR_INFO procedure may result in unpredictable processor behavior

Problem: If a PMI is taken during the execution of the PAL_MC_ERROR_INFO procedure, the branch return information stored by the PAL call may be lost. As a result, the behavior of the processor is not guaranteed upon its return from the PMI handler.

Implication: PAL_MC_ERROR_INFO may not complete successfully and the processor behavior may be unpredictable.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

86. Data-poisoning bits not included in PAL_MC_ERROR_INFO cache_check and bus_check structures

Problem: In the *Intel® Itanium® Architecture Software Developer's Manual Specification Update* machine check architecture extensions were added for supporting data-poisoning events. These extensions will help in supporting different data-poisoning handling policies. Current Itanium 2 processors do not implement the dp bit in the cache_check and bus_check structures in PAL_MC_ERROR_INFO.

Implication: When parsing error logs, the OS cannot distinguish between some hardware generated corrected events versus data-poisoning events.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

87. PAL_PREFETCH_VISIBILITY call not implemented

Problem: Calling PAL_PREFETCH_VISIBILITY with trans_type argument of 1 returns Invalid Argument.

Implication: PAL_PREFETCH_VISIBILITY does not support physical addressing attribute transitions.

Workaround: None at this time.

Status: For the steppings effected, see the Summary Table of Changes.

Errata (IA-32 Execution Layer)

1. Ordering of loads and stores

Problem: IA-32 execution layer reorders IA-32 loads and stores during code optimization. Under some conditions, IA-32 applications sharing memory between processes executing on IA-32 execution layer may not maintain processor ordering of loads and stores.

Implication: Multiprocessor or multi-threaded IA-32 applications that share memory between processes and depend upon processor ordering may not behave as expected. Locks, semaphores, and all other fencing instructions maintain strong ordering and have no exposure to this erratum. Intel has not been able to reproduce incorrect program behavior due to this erratum with commercial software.

Workaround: Multiprocessor or multi-threaded IA-32 applications should protect access to shared variables with locks, semaphores, or OS synchronization.

Status: For the versions affected, see the Summary Table of Changes.

2. Segmentation not supported

Problem: IA-32 execution layer does not support segmentation, and only limited support for segmentation registers is provided.

Implication: IA-32 applications that use segmentation may not operate as expected when executing on IA-32 execution layer. Check with your OS vendor to determine if segmented IA-32 applications are supported.

Workaround: IA-32 applications should use the flat 32-bit addressing.

Status: For the versions affected, see the Summary Table of Changes.

3. 16-bit application mode not supported

Problem: IA-32 execution layer does not support 16-bit application mode. The size address prefix (0x67) is supported only for allowed segment overrides.

Implication: IA-32 applications running on IA-32 execution layer that use 16-bit application mode may not behave as expected. IA-32 execution layer does support 16-bit instructions.

Workaround: IA-32 applications should use 32-bit application mode.

Status: For the versions affected, see the Summary Table of Changes.

4. IA-32 floating-point state

Problem: FPUDataPointer, FPUInstructionPointer, and FPULastInstructionOpcode fields of the floating-point (FP) state are not updated by the `FSAVE`, `FNSAVE`, `FXSAVE`, `FSTENV`, and `FNSTENV` instructions.

Implication: IA-32 code running on IA-32 execution layer using `FSAVE`, `FNSAVE`, `FXSAVE`, `FSTENV`, or `FNSTENV` instructions cannot retrieve FPUDataPointer, FPUInstructionPointer, and FPULastInstructionOpcode fields from the last non-control FP instruction using these instructions. The last FP state is guaranteed only upon unmasked FP exceptions.

Workaround: To get FP state on exceptions, one needs to use the OS-provided context. For example, the user can get the exception record from Windows* or use sigcontext on Linux*.

Status: For the versions affected, see the Summary Table of Changes.

5. Floating-point C1 condition code flag support

Problem: IA-32 execution layer does not set the floating-point C1 condition code flag when the last rounding by the instruction was upward. Other C1 behavior is unaffected.

Implication: IA-32 code running on IA-32 execution layer that depends upon the C1 condition code flag to identify upward rounding may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

6. IA-32 floating-point pseudo-denormal, pseudo-NaN, and pseudo-infinity support

Problem: IA-32 execution layer will treat pseudo-denormal, pseudo-NaN, and pseudo-infinity values as un-normalized numbers, normalize them, and continue operation rather than raise a denormal exception.

Implication: IA-32 code running on IA-32 execution layer using pseudo-denormal, pseudo-NaN, and pseudo-infinity values may not behave as expected. Note that IA-32 processors since the Intel® 387 math coprocessor do not generate pseudo-denormal, pseudo-NaN, and pseudo-infinity values.

Workaround: IA-32 applications should avoid using floating-point encodings not supported by the final version of the IEEE Standard 754.

Status: For the versions affected, see the Summary Table of Changes.

7. Behavior of quiet and signaling NaNs

These NaN operations have the following behavior:

1. Floating-point operations involving an SNaN operand and a QNaN operand will return a QNaN with the significand of the lesser operand. When moving values using `FLD` followed by `FSTP`, IA-32 execution layer may not convert SNaNs to QNaNs.
2. SSE operations performed on a pair of XMM registers that contain QNaN values may result in the destination changing to the resultant QNaN.

Implication: IA-32 code running on IA-32 execution layer that depends upon SNaN or QNaN behavior may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

8. IA-32 floating-point exceptions

Problem: On a FP exception, IA-32 execution layer will set the denormalized operand exception flag when a denormal value has been stored and will set the inexact precision exception flag when an unmasked overflow/underflow fault occurs.

Implication: IA-32 code running on IA-32 execution layer depending upon the denormalized or inexact precision flags may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

9. Partial support for EFLAGS

Problem: IA-32 execution layer supports the ID, OF, DF, SF, ZF, AF, PF, CF, and TF EFLAG bits. The IF flag is held to 1. The VIP, VM, and IOPL flags are held to 0. The AC, NT, and RF flags can be written and read by POPF and PUSHF operations, but their semantics are not simulated.

Implication: IA-32 code running on IA-32 execution layer depending upon privileged EFLAGS state or the AC, NT and RF flags may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

10. EFLAGS and floating-point exception flag behavior

Problem: EFLAG and FP exception flags may have incorrect behavior when read from an exception handler context, when read from another thread or process, or read by self-modifying code if the flags are not consumed in the original context.

Note: EFLAG and FP exception flags are correct under the use of a debugger.

Implication: Multiprocess, multi-threaded, or self-modifying IA-32 code running on IA-32 execution layer reading EFLAGS or FP exception flags may not behave as expected if the flags are not consumed in the original context.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

11. RSM and IRET instructions raise incorrect faults

Problem: On IA-32 execution layer, `RSM` calls raise a general protection fault, and `IRET` calls raise an illegal operation fault.

Implication: These are not expected to occur in user mode.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

12. Cross-modifying code

Problem: IA-32 execution layer may not maintain execution consistency of multiprocess cross-modifying IA-32 code if a process has opened the instruction page with read-only permission.

Implication: Multiprocess cross-modifying IA-32 applications may not behave as expected, if a process has opened the instruction page with read-only permission.

Workaround: Multiprocess cross-modifying IA-32 applications should open modified instruction pages with read/write access.

Status: For the versions affected, see the Summary Table of Changes.

13. Atomicity of lock-prefixed instructions making unaligned memory references

Problem: On IA-32 execution layer, an IA-32 lock-prefixed instruction making an unaligned memory reference is performed atomically only with respect to other lock-prefixed instructions making unaligned memory accesses in the same process.

Implication: If an unaligned memory access is made to the same physical address by a lock-prefixed instruction and another process, an instruction without a lock prefix, or an aligned lock-prefixed instruction, atomicity is not guaranteed, and the code may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

14. Atomicity of lock-prefixed instructions making uncacheable memory references

Problem: On IA-32 execution layer, an IA-32 lock-prefixed instruction making an uncacheable memory reference is performed atomically only with respect to other lock-prefixed instructions making uncacheable memory accesses in the same process.

Implication: If an uncacheable memory access is made to the same physical address by a lock-prefixed instruction and another process, an instruction without a lock prefix, or an uncached lock-prefixed instruction, atomicity is not guaranteed and the code may not behave as expected.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

15. Noninterruptability of 32-bit unaligned and 16-byte stores

Problem: On IA-32 execution layer, if a thread is suspended during a 32-bit unaligned or a 16-byte IA-32 store to cached memory, another thread may observe partially updated memory until the OS can service the thread suspension.

Implication: When a process performs 32-bit unaligned or 16-byte stores, partial memory updates may be observed by other threads until the OS can service the thread suspension, resulting in unexpected behavior.

Workaround: None at this time.

Status: For the versions affected, see the Summary Table of Changes.

16. IA-32 execution layer install and uninstall failures

Problem: On some Itanium 2-based platforms, incorrect reports may be seen while installing or uninstalling IA-32 execution layer.

Implication: During installation, the IA-32 execution layer installer “IA-32ExecutionLayerSetup.exe” may incorrectly report that a previous version has been installed and ask the user to remove the previous installation.

After an uninstall and subsequent reboot, the system may incorrectly ask users to reinstall IA-32 execution layer.

Workaround: Users should download the latest IA-32 execution layer installer “IA-32ExecutionLayerSetup_1.exe”(revision 1 or greater) from the Microsoft* download center.

Status: For the versions affected, see the Summary Table of Changes.

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Changes

There are no Specification Changes for this revision of the *Intel® Itanium® 2 Processor Specification Update*.

Itanium® 2 Processor (up to 3 MB L3 Cache) Specification Clarifications

There are no Specification Clarifications for this revision of the *Intel® Itanium® 2 Processor Specification Update.*

Itanium® 2 Processor (up to 3 MB L3 Cache) Documentation Changes

There are no Documentation Changes for this revision of the *Intel® Itanium® 2 Processor Specification Update.*

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Changes

There are no Specification Changes for this revision of the *Intel® Itanium® 2 Processor Specification Update.*

Itanium® 2 Processor (up to 6 MB L3 Cache) Specification Clarifications

There are no Specification Clarifications for this revision of the *Intel® Itanium® 2 Processor Specification Update*.

Itanium® 2 Processor (up to 6 MB L3 Cache) Documentation Changes

There are no Documentation Changes for this revision of the *Intel® Itanium® 2 Processor Specification Update.*

IA-32 Execution Layer Specification Clarifications

1. Aliasing of MMX registers to FP registers

If a value is written to the FP register, and an MMX™ operation is performed to the corresponding MMX register, the exponent portion of the corresponding FP register may not be written to 1's if the register's significand is unchanged by the MMX instruction.

As described in the *IA-32 Intel® Architecture Software Developer's Manual*, the EMMS instruction, which empties the MMX state by setting the tags in the x87 FPU tag word to 11B, must be executed at the end of an MMX routine before calling other routines that can execute FP instructions

2. Floating-point and SSE precision

Floating-point and SSE instructions like RCPPS, RCPSS, RSQRTPS, and RSQRTSS may provide slightly more precise results than Itanium 2 processors or IA-32 Intel processors since IA-32 execution layer may merge separate FADD and FMUL instructions into a single FMA instruction or replace two roundings by one rounding.

3. CPUID values represent the IA-32 execution layer processor model

CPUID return values accurately represent the IA-32 execution layer processor model, but may not represent the physical processor in the system. The vendor and family information are correct for IA-32 execution layer, but cache, translation lookaside buffer (TLB), and other processor-specific information is not supported. The CPUID values returned by IA-32 execution layer will be documented in the *Intel® Processor Identification and the CPUID Instruction Application Note* (AP-485).

4. IA-32 execution layer resides in the application virtual address space

IA-32 execution layer components, memory for translated code blocks, and IA-32 execution layer data structures, all reside in the application virtual address space. Memory requests may be denied if insufficient memory is available. Non-relocatable DLLs may fail to load if that memory is already occupied.

5. Signal delivery may be postponed during code translation or garbage collection

During code translation or garbage collection, signal delivery may be postponed. There is no maximum time-limit, but the delivery is guaranteed to happen eventually.

6. Aborting threads could cause other process threads to hang

An application running on IA-32 execution layer may use internal IA-32 execution layer critical objects. Aborting a thread that holds an IA-32 execution layer critical object could cause the other threads in the process to hang.

7. Core dump files cannot be produced correctly when an IA-32 process is aborted

When an IA-32 process is aborted, a core dump file can often be used for debugging purposes. Unfortunately, at this time, the core dump files created from an aborted process using IA-32 execution layer does not contain valid information.

8. The I/O Privilege Level (IOPL) mechanism is not implemented

The I/O Privilege Level (IOPL) mechanism is not implemented and is hard coded to 0. As a result, all applications that use the IN or OUT instructions, as well as CLI and STI, will result in a #GP fault.

9. Software interrupts must be supported by the OS

Software interrupts (`INT` instructions) are only implemented to the extent that they are supported by the OS, by converting them into an Itanium exception.

10. Intersegment calls require OS mechanism

`FAR CALL`, `FAR JMP`, `FAR RET`, `SYSENTER`, and `SYSEXIT` instructions are supported only when there is a standard interface mechanism in the OS. Call gates and hardware task switch mechanisms are not supported.

11. Thread creation may be reported incorrectly to the OS

Thread creation may succeed according to the OS, but could later fail inside IA-32 execution layer due to insufficient resources (memory/handle/semaphore). The created thread will never start running.